DU[illegible]X LIBRARY
NA[illegible]RADUATE SCHOOL
MONTEREY CA 93943-5101

Unclassified
ECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

a. REPORT SECURITY CLASSIFICATION NCLASSIFIED	1b. RESTRICTIVE MARKINGS
a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
a. NAME OF PERFORMING ORGANIZATION laval Postgraduate School	6b. OFFICE SYMBOL OR
c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code)
a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL
c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

1. TITLE (Including Security Classification)
A MULTILEVEL APPROACH TO MINIMUM COST NETWORK FLOWS

2 PERSONAL AUTHOR(S)
CAVANAUGH, Kevin J.

3 TYPE OF REPORT	13b. TIME COVERED FROM TO	14. DATE OF REPORT (Year, Month, Day)	15. Page Count
Master's thesis		1992, SEPTEMBER	105

6. SUPPLEMENTAL NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

7. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Multigrid, Multilevel, Optimization, Networks, Transportation Problem, Aggregation

9. ABSTRACT (Continue on reverse if necessary and identify by block number)

Multigrid/multilevel techniques were originally developed for numerically solving partial differential equations. This thesis explores the application of multilevel techniques to a non-geometric long transportation problem. An introduction to multigrid is given, and specifics of how it is applied to this minimum cost network flow problem are explored.

This research shows that multilevel techniques can be applied to network optimization problems. Further, since a previous restriction is removed by transferring the problem from a physical space to a cost space, the techniques can be applied to a broader range of problems.

Both a multilevel V-cycle and a Full Multigrid (FMG) algorithm are implemented. Various strategies for restriction and local relaxation are discussed, and comparisons between the methods are made. Directions for future work include investigation of graph theoretic aspects of the problems, implementation of a regular grid overlay of the domain, exploration of a fast adaptive composite (FAC) grid algorithm, and development of a full approximation scheme (FAS) algorithm.

) DISTRIBUTION/AVAILABILTIY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC	1a. REPORT SECURITY CLASSIFICATION Unclassified
2a. NAME OF RESPONSIBLE INDIVIDUAL an Emden Henson	22b. TELEPHONE (Include Area Code) (408)646-2198 / 22c. OFFICE SYMBOL MA/Hv

DD Form 1473, JUN 86 — Previous editions are obsolete. S/N 0102-LF-014-6603 — SECURITY CLASSIFICATION OF THIS PAGE Unclassified

T259858

Approved for public release; distribution is unlimited.

A Multilevel Approach to Minimal Cost Network Flows

by

Kevin J. Cavanaugh
Lieutenant Commander, United States Coast Guard
B.S., United States Coast Guard Academy

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS ANALYSIS
MASTER OF SCIENCE IN APPLIED MATHEMATICS
from the

NAVAL POSTGRADUATE SCHOOL

September, 1992

ABSTRACT

This thesis presents an exploration of the application of multigrid/multilevel techniques to a non-geometric long transportation problem. An introduction to multigrid is given, and specifics of how it is applied to this minimum cost network flow problem are explored.

This research shows that multilevel techniques can be applied to network optimization problems. Further, since a previous restriction is removed by transferring the problem from a physical space to a cost space, the techniques can be applied to a broader range of problems.

Both a multilevel V-cycle and a Full Multigrid (FMG) algorithm are implemented. Various strategies for restriction and local relaxation are discussed, and comparisons between the methods are made. Experimental results are given. Directions for future work include investigation of graph theoretic aspects of the problem, implementation of a regular grid overlay of the domain, exploration of a fast adaptive composite (FAC) grid algorithm, and development of a full approximation scheme (FAS) algorithm.

Thesis
C338242
C 1

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

TABLE OF CONTENTS

I. INTRODUCTION 1

A. NETWORK OPTIMIZATION 1

B. MULTIGRID AND MULTILEVEL TECHNIQUES 2

C. THESIS OVERVIEW 4

II. TRADITIONAL METHODS 5

A. SIMPLEX ALGORITHM 5

B. PRIMAL NETWORK SIMPLEX METHOD 9

III. AN INTRODUCTION TO MULTILEVEL METHODS 13

A. PDE EXAMPLE 15

B. COARSE GRID CORRECTION AND THE MULTIGRID V-CYCLE 18

C. NESTED ITERATION AND THE FULL MULTIGRID (FMG) CYCLE 24

D. THE FULL APPROXIMATION SCHEME (FAS) 25

E. MULTIGRID APPROACH TO THE TRANSPORTATION PROBLEM 28

IV. DESIGN AND IMPLEMENTATION OF THE V-CYCLE 30

A. FIRST PRELIMINARY DESIGN PHASE 30

B. RESTRICTION 36

C. INTERPOLATION 39
D. NESTED ITERATION 45
V. THE FULL MULTIGRID ALGORITHM 49
A. VARIATION OF COARSENING SCHEMES 50
1. Flow and Demand Weighting 50
2. Cost Conversion Using Dual Multipliers 52
B. LOCAL RELAXATION 55
1. Local Relaxation by Optimization 55
2. Local Relaxation by Cycle Removal 62
VI. CONCLUSIONS AND RECOMMENDATIONS 71
A. SIGNIFICANT RESULTS 72
B. AVENUES OF FURTHER RESEARCH 73
APPENDIX 76
LIST OF REFERENCES 93
INITIAL DISTRIBUTION LIST 94

LIST OF TABLES

Table I. **COST SPACE VS. PHYSICAL SPACE** 32

Table II. **COMBINATIONS OF SUBPROBLEM SIZE AND OVERLAP** ... 61

Table III. **CYCLE REMOVAL ALGORITHM** 70

LIST OF FIGURES

Figure 1. **A Small Transportation Problem with Flows** 7

Figure 2. **High Frequency Wave Appears Low Frequency on Coarser Grid** 18

Figure 3. **Multigrid V-Cycle** 23

Figure 4. **Full Multigrid (FMG) V-Cycle** 25

Figure 5. **Cost Space vs. Physical Space** 31

Figure 6. **Transformation to Reduced Dimension Space** 35

Figure 7. **Coarsening of Demand Nodes** 37

Figure 8. **Five by Two Transportation Problem: Illustration of $M\times 2$ Heuristic** 41

Figure 9. **Computation of Dual Multipliers** 54

Figure 10. **Comparison of Initial Flow and Optimal Flow** 57

Figure 11. **Four Node Relaxation with Single Node Overlap** 60

Figure 12. **Unions of Various Two Supply Node Problems** 64

Figure 13. **Unions of Three Supply Node Subproblems.** 65

Figure 14. **Cycle Removal** 68

ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance, support, supervision and tolerance of many people, and I would like to take this opportunity to thank them.

I am sincerely grateful to Drs. Tom Manteuffel and Steve McCormick of the University of Colorado, Denver, for sponsoring my attendance at the 1992 Conference on Iterative Methods. I would like to thank Dr. Achi Brandt for taking precious time out of his hectic schedule, both at the Copper Mountain Conference and during his visit to Monterey, to provide me with assistance and ideas. As usual, his suggestions were enormously helpful.

I deeply appreciate the assistance of Dr. Gordon Bradley in developing a subroutine version of GNET for me to experiment with, and the time he spent in helping me understand the program. I would also like to thank Dr. Richard Rosenthal for his assistance. His guidance greatly improved both the writing of the thesis and the thesis presentation.

My very deep thanks go to Dr. Van Emden Henson, for struggling along with a thesis student who frequently failed to take his advice. I will always be indebted to him for removing my PDE anxiety, and for teaching me that Fourier transforms are only rainbows.

Finally, I want to express my deepest thanks to my wife, Claire, who took care of the really important things while I worked on this thesis. I could not have done this without her love and support, and for both of these I will be eternally grateful.

I. INTRODUCTION

A. NETWORK OPTIMIZATION

Optimization problems occur frequently in the areas of transportation, manpower planning, industrial engineering, production, resource allocation, and many others. Optimization problems of the form

$$\begin{aligned} &\textit{Minimize } \mathbf{c}^T\mathbf{x} \\ &\textit{Subject to: } \mathbf{Ax}=\mathbf{b} \end{aligned}$$

are known as linear programming problems (Nemhauser and Wolsey, 1988). Minimum cost network flow problems are a specialization of linear programming optimization problems, characterized by having a one—to—one correspondence with a digraph consisting of a set of nodes or vertices, and a set of directed arcs or edges joining pairs of nodes.

Many problems arise in connection with network flows. They include finding the shortest path between any two nodes, finding the minimum cost flow which meets all constraints, determining the maximum amount of flow that a capacitated network can accommodate, finding a path which visits all nodes in the network exactly once (a Hamiltonian circuit), or finding the shortest such path (the classical traveling salesman problem).

Some network flow problems, such as the traveling salesman problem, are NP-complete. That is, they belong to a class of hard problems for which no polynomial time algorithms are known. Polynomial time algorithms exist for solving some of the others, such as the shortest path problem, depending on the parameters of the problem. The shortest path problem with mixed sign arc lengths, for example, is NP-complete. (Bazaraa et. al., 1990). The transportation problem, which is addressed in this thesis, is not NP-complete, however, in developing the application of a technique such as multigrid to a new class of problems, working on a problem such as this is a necessary first step before proceeding to more difficult ones.

The list of applications of network flow problems is quite extensive. Military applications range from scheduling ships for underway replenishment, to interdiction of precursor chemicals used in illegal drug production. Civilian applications include scheduling oil tanker deliveries, airline crew schedules, and personnel assignment models, among many others. The interest in network problems, and the potential benefits of improved solution methods, are enormous.

B. MULTIGRID AND MULTILEVEL TECHNIQUES

Multigrid and multilevel techniques were originally developed for solving partial differential equations numerically and have been applied successfully to a growing number of diverse problems. In general terms, these techniques work best on problems where traditional iterative methods show great initial improvement, but later stall. Achi Brandt, one of the developers of multigrid, said that "stalling numerical processes must

be wrong" (Brandt, 1984). Multigrid methods are most effective when local information must be propagated globally. That is, when a local operation, such as a partial derivative or a change in flow on an arc, has global impact on the problem, there is potential benefit from a multilevel approach.

There are some superficial similarities between the partial differential equations for which multigrid was originally developed and network flow problems. Changing the flow on an arc, like taking a partial derivative, is a very local operation. It is performed on that arc alone. The arc, however, like a point in the domain of the PDE, does not exist in a vacuum, and changing the flow on that arc will require changes on many other arcs in the problem. Multigrid excels at spreading this local information throughout the domain of the problem. Because of these similarities, and because of the potential gain from multigrid, this approach is worth researching.

Some work has been done in applying a multigrid approach to optimization and network flow problems, mostly at the Weizmann Institute of Science in Rehovot, Israel (Brandt, Ron, and Amit, 1985; Ron, 1987). Of particular interest is a master's thesis on a multilevel approach to the long transportation problem (Kaminsky, 1989). This work provided background and a starting point for this thesis, and is described briefly in Chapter III.

The goal of this thesis is to expand on the work done at the Weizmann Institute in applying multilevel techniques to optimization by removing one of the restrictions placed on the long transportation problem by Kaminsky (1989). Also, a comparison of multilevel techniques to more traditional network optimization methods is made.

C. THESIS OVERVIEW

The organization of this thesis is as follows: Chapter II will define the transportation problem and address the traditional methods for solving it, including the simplex algorithm and a network simplex algorithm. Chapter III will give a brief introduction to multigrid and multilevel methods, including the background of the techniques from their origins in partial differential equations, and a discussion of some previous work in applying multilevel methods to the transportation problem. Chapter III will also introduce some notational conventions used in discussing multilevel methods.

Chapter IV discusses the first phase of the current research. This chapter includes the basic building blocks of a multigrid algorithm, the specific relaxation and interpolation strategies and how they are applied to the transportation problem. Also, this chapter will describe how one of the restrictions which was necessary in previous work, was removed. The removal of this restriction allows the application of the multilevel techniques to a large class of problems that could not previously be treated by such methods.

Chapter V is concerned with more advanced multilevel techniques and questions, including the application of a full multigrid (FMG) algorithm and experimentation with several local relaxation methods. Some new coarsening strategies are also discussed. Chapter VI contains conclusions and recommendations for further research, and the FORTRAN code implementing the algorithm is contained in the Appendix.

II. TRADITIONAL METHODS

A. SIMPLEX ALGORITHM

The transportation problem is among the simplest of network flow problems. It is posed on a bipartite graph, consisting of a set of M supply nodes, a set of N demand nodes, and a set of arcs each connecting a supply node to a demand node. Each supply node i has a fixed amount a_i of a commodity which it can provide. Each demand node j has a fixed requirement b_j for that commodity, and each arc (i,j) connecting supply node i to demand node j has a cost per unit flow c_{ij} associated with it. The transportation problem has been called the 'white laboratory rat' of network optimization problems. Because of the simplicity of its structure, new algorithms are first tested on it before being applied to more complicated problems.

If the total supply equals the total demand, the problem is *balanced.* An unbalanced problem can be transformed into a balanced problem by the addition of a dummy node. When $M<<N$, the problem is referred to as a *long transportation problem.* Denoting the flow on arc (i,j) by x_{ij}, the transportation problem can be expressed as: given $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$, find $\boldsymbol{x}$ to

$$\begin{aligned} &\text{Minimize } z=\sum_{i=1}^{M}\sum_{j=1}^{N}c_{ij}x_{ij} \\ &\text{Subject to: } \sum_{j=1}^{N}x_{ij}=a_i \quad \forall i \\ &\qquad\qquad \sum_{i=1}^{M}x_{ij}=b_j \quad \forall j \\ &\qquad\qquad \boldsymbol{x}\geq\boldsymbol{0} \end{aligned}$$

or, in matrix notation,

$$\begin{aligned} &\text{Minimize } z = \mathbf{c}^T\mathbf{x} \\ &\text{Subject to: } \mathbf{A}\mathbf{x} = \mathbf{b} \\ &\qquad\qquad\quad \mathbf{x} \geq \mathbf{0} \end{aligned}$$

where $\mathbf{b}$ is an $(M+N)$-vector whose first M entries are the available supplies at nodes S_1 through S_M, and whose last N entries are the required demands at demand nodes D_1 through D_N. Let K be the number of arcs in the problem. The K-vector $\mathbf{x}$ is composed of the flow on the arcs, and the vector $\mathbf{c}$ is the cost of those arcs. The matrix $\mathbf{A}$ has as many rows as there are nodes in the problem, $M+N$, and as many columns as there are arcs. Each column of $\mathbf{A}$ is associated with an arc of the problem, and they are arranged in an order that matches the order of $\mathbf{c}$ and $\mathbf{x}$. Each column has exactly two non-zero entries: a +1 in the row corresponding to the supply node of the arc, and a -1 in the row corresponding to the demand node. Each row of $\mathbf{A}$ is associated with one of the constraints of the problem (Bazaraa, et.al., 1990). A small example problem is shown in Figure 1, for which

$$\mathbf{x} = \begin{bmatrix} 3 \\ 5 \\ 4 \\ 3 \\ 10 \\ 2 \\ 2 \\ 5 \\ 3 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 2 \\ 1 \\ 5 \\ 6 \\ 4 \\ 3 \\ 1 \\ 0 \\ 4 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 12 \\ 15 \\ 10 \\ -8 \\ -20 \\ -9 \end{bmatrix}, \quad \text{and } \mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ -1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 \end{bmatrix}.$$

Note that $\mathbf{x}$ in this case is only one of many possible flow combinations, and does not represent an optimal solution.

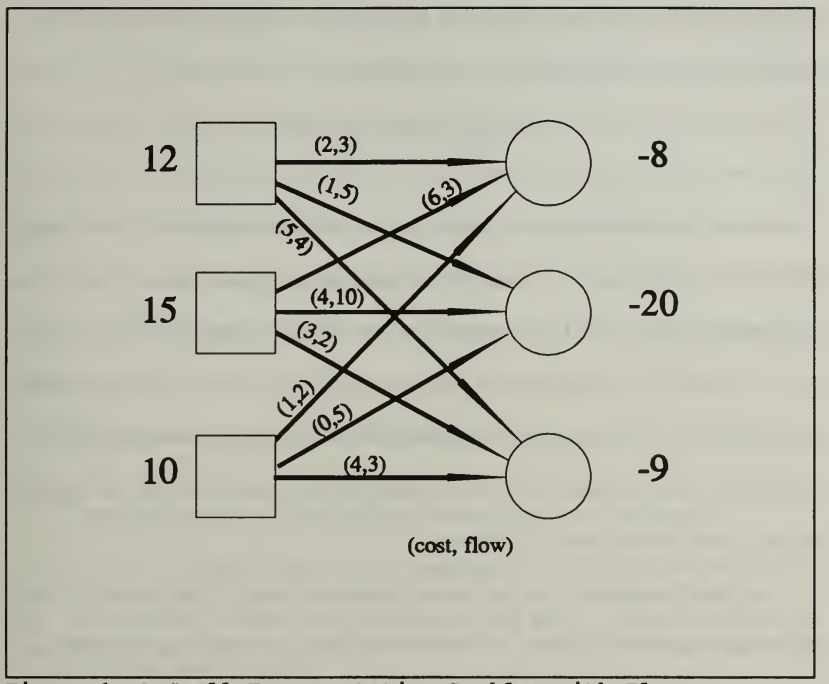


Figure 1. A Small Transportation Problem with Flows

Since the matrix A is rank-deficient (Nemhauser and Wolsey, 1988), an artificial or root arc is added to the problem by augmenting A with $M+N$ standard basis vectors e_1 through e_{M+N}. The new matrix A can be partitioned $A = [B\ N]$, rearranging columns if necessary, where B is a square matrix, ($M+N$ by $M+N$), with linearly independent columns, referred to as a *basis* or *basic matrix*. Since B has full rank, there exists a transformation matrix Z such that $BZ = N$.

Associated with every basis B is a unique vector $\hat{x}$, such that $B\hat{x}=b$. Appending to this an appropriate size zero vector creates a *basic solution*, x^0. If all elements of x^0

are greater than or equal to zero, x^0 is a *basic feasible solution.* Any solution which satisfies the constraints, $Ax=b$, can be rewritten as

$$[B\ N]\begin{bmatrix} x_B \\ x_N \end{bmatrix} = Bx_B + Nx_N = b.$$

A basic feasible solution corresponds to an extreme point of the feasible region defined by $Ax=b$. Since the transportation problem is a specialization of the linear programming problem, if a finite optimum solution exists, one will occur at an extreme point. In network flow problems, arcs which have positive flow at an extreme point are associated with a corresponding x_B. The $M+N$ arcs in x_B (including the arcs to the root), constitute a rooted spanning tree for the graph, called a basis tree, and the optimal solution is the spanning tree with least cost.

Any feasible solution x can be written in terms of a basic feasible solution, $\hat{x}$, and the nonbasic variables x_N. Since $N = BZ$, $Nx_N = BZx_N$, by substitution $Bx_B+BZx_N=b$. Solving for x_B gives $x_B = B^{-1}b-Zx_N=\hat{x}-Zx_N$, since $\hat{x}=B^{-1}b$.

Partitioning the cost vector c to conform with A, i.e., $c=[c_B^T\ c_N^T]^T$, the objective function value for the current solution x can be expressed as

$$\begin{aligned} c^T x &= [c_B^T\ c_N^T]\ [x_B\ x_N]^T \\ &= c_B^T x_B + c_N^T x_N \\ &= c_B^T(\hat{x}-Zx_N) + c_N^T x_N \\ &= c_B^T\hat{x} - c_B^T Z x_N + c_N^T x_N \\ &= c_B^T\hat{x} + (c_N^T - c_B^T Z)\, x_N \\ &= c_B^T\hat{x} + (c_N^T - u^T N)\, x_N \end{aligned}$$

where u, sometimes referred to as the vector of dual multipliers, is the solution to the equation $u^T B = c_B^T$.

Since x_N equals $\mathbf{0}$ at an extreme point, flow on the non-basic arcs can only be increased. This means that the objective function value of the current solution can be improved only if there is a column of N, say N_k, such that $c_k - u^T N_k < 0$, where c_k is the k th entry of c_N. If such a column exists, then the current extreme point may not be the optimal one, and an improved solution may be found by moving to an adjacent extreme point. This is accomplished by bringing the non-basic arc corresponding to N_k into the basis, and removing one of the arcs currently in the basis. This operation is known as *pivoting*.

The simplex method can be summarized as follows:

1. Start from an basic feasible solution, using artificial arcs if necessary.
2. *Price out* non-basic variables. By computing $c_k - u^T N_k$, the *reduced cost* of arc k.
3. If an variable is found with a reduced cost less than 0, perform a ratio test to determine which variable will leave the basis, pivot the new variable into the basis, and return to step 2.
4. If no non-basic variables have a reduced cost less than 0, the current solution is optimal.

B. PRIMAL NETWORK SIMPLEX METHOD

A traditional method for solving the transportation problem is a primal network simplex algorithm. In this approach, an initial basic feasible solution is found by starting with an artificial basis. That is, an artificial node and an arc from every real node to this artificial node are added to the problem. These artificial arcs are used as an initial basis, with an extremely large cost associated with each arc. This cost is large enough that the

algorithm will try to remove these arcs from the basis as quickly as possible. Once an artificial arc is removed from the basis, it is removed from the problem. Once all artificial arcs have zero flow, feasibility has been achieved.

One example of a network simplex algorithm is GNET, developed by Bradley, Brown and Graves (1977). A brief introduction to GNET is in order at this point for two reasons. First, an understanding of the traditional approach exemplified by GNET will help to clarify the differences in a multilevel approach. Second, during the first part of the current research, multilevel techniques were used as a way of producing a feasible starting solution for GNET, rather than the artificial basis which is normally used.

GNET uses a set of extremely efficient data structures to represent the necessary information about the problem. Since in most problems the number of arcs greatly exceeds the number of nodes, storing the graph in the form of an explicit arc list, or an adjacency matrix, is very wasteful. GNET uses a reverse star storage scheme to limit the amount of storage required. Three K-length arrays and two $(M+N)$-length array are used to describe the network. The K-length arrays are **T**, which stores the tail node of each arc; **C**, which stores the associated cost, **CP**, which stores the associated capacity. Supplies and demands are stored in an $M+N$ length vector **X**. Arcs with a common head node are stored contiguously, and the entry point for each group of arcs is stored in **H**, the $(M+N+1)$-length head array.

The network simplex algorithm as implemented in GNET works on the spanning tree defined by the basic arcs. The price-out step is performed by comparing the cost of each non-basic arc (i,j) to the difference between the duals of the supply node and the

demand node it connects. Call these duals u_i and u_j. Because of the structure of the matrix N, $\boldsymbol{u}^T\boldsymbol{N}_k$ is always equal to u_i-u_j, where N_k represents the arc from supply node i to demand node j. Therefore, the reduced cost of arc (i,j) is c_{ij}-$(u_i$-$u_j)$. If this arc prices out favorably, that is, if the reduced cost is less than zero, it is considered for entering the basis.

The introduction of any new arc (i,j) into the basis tree will create a cycle consisting of (i,j) and the nodes on the paths from i and j towards the root, until the two paths reach a common node, called the *join*. Duals of some of the successors of the join will change after the pivot, but the flow will change only on the arcs within the cycle. Specifically, flow will increase on those arcs pointing in the same direction around the cycle as the new arc, and decrease on arcs which point in the opposite direction.

The primal network simplex algorithm is, then

1. Start from a basic solution, using artificial arcs if necessary.

2. Compute the duals for all nodes in the problem with the current solution. That is, solve $\boldsymbol{u}^T\boldsymbol{B} = \boldsymbol{c}_B{}^T$. Since $\boldsymbol{B}$ can be made triangular through elementary row operations, this equation can be solved by starting at the root node and working out towards the leaf nodes.

3. Price out non-basic arcs to compute the reduced costs, c_{ij}-$(u_i$-$u_j)$. If any reduced cost is negative, pivot an arc with negative reduced cost into the basis and return to step 2.

4. If the reduced cost of all non-basic arcs is greater than 0, current solution is optimal.

GNET does not strictly follow the above procedure. After a pivot, new dual multiplier values are computed through a more efficient update process. For more details concerning the working of GNET, see Bradley, Brown and Graves (1977).

III. AN INTRODUCTION TO MULTILEVEL METHODS

Much of the theory of multigrid and multilevel methods comes from their development as methods for solving partial differential equations (PDEs). Applications of multilevel methods have expanded beyond their genesis in PDEs, however, and understanding the roots of multigrid provides insight into the reasons behind taking the multilevel approach.

One method of solving partial differential equations numerically is to discretize the problem over a finite number of discrete points in the domain, and then to approximate the continuous derivatives in the PDE by discrete finite differences. This leads to a linear system of many equations in many unknowns. Since the finite difference approximations use only the values from near neighbors of a point, the resulting matrix equations are banded and extremely sparse.

To solve this system of equations, $\boldsymbol{Ax} = \boldsymbol{b}$, one approach is to use some sort of iterative process $\boldsymbol{x}^{(new)} = \boldsymbol{Gx}^{k}$, where $\boldsymbol{G}$ is an iteration matrix designed to start at an initial estimate, say $\boldsymbol{x}_0$, and ultimately converge to the exact solution, $\boldsymbol{x}$. After several iterations, an approximation, $\boldsymbol{x}^{(new)}$, to the exact solution is reached.

There are two measures for how good an approximation $\boldsymbol{x}^{(new)}$ is. The first is the *error*, which is the difference between $\boldsymbol{x}$ and $\boldsymbol{x}^{(new)}$; that is, the difference between the exact solution and the approximation. In practice, however, the exact solution is not known and so the error is as difficult to determine as the solution. The second measure

is called the *residual*, $r^{(new)} = b - Ax^{(new)}$, which is a measure of by how much the approximation fails to satisfy the system of equations. This is a much more useful measure in practice.

Any vector on a grid with N-1 internal gridpoints can be represented as the sum of N+1 vectors $v_j = \sin(jk\pi/N)$, $j=0,1,...,N$, $k=1,2,...,N-1$, where j is the associated gridpoint of v, and k, the *wavenumber* of the mode, is the number of half sine waves in v over the domain of the problem. These vectors, the *Fourier modes*, form a basis for the vector space. That is, any vector, including the error vector, can be expressed as a linear combination of the Fourier modes. A Fourier mode with a low wavenumber will consist of long, smooth waves. A large wavenumber indicates a highly oscillatory wave. Note that the number of modes necessary to span the domain of the problem is dependent on the number of gridpoints (Briggs, 1987).

If we call the error e, it follows that

$$\begin{aligned} Ae^{(new)} &= A(x - x^{(new)}) \\ &= Ax - Ax^{(new)} \\ &= b - Ax^{(new)} \\ &= r^{(new)} \end{aligned}$$

In theory, if the error vector could be found from this equation, it could be added to the approximate solution to yield the exact solution to the original system. Again, in practice the benefit from this observation is of limited use. The residual equation, as this equation is known, is just as difficult to solve as the original equation.

This does not mean, however, that the residual equation is useless. While we may know very little about the characteristics of the solution to the original problem, we know

that after a few cycles of an iterative process, for many problems, the error of an approximation will be smooth. As will be demonstrated, this property is critical in a multilevel approach to solving this system of equations (Briggs, 1987).

A. PDE EXAMPLE

A model problem will help to illustrate the multigrid algorithm and the advantages gained from a multigrid approach. The model problem is a second order differential equation boundary value problem on the unit interval in one dimension, $-u''(x)+\sigma u(x)=f(x),\ 0<x<1$ ($\sigma \geq 0$ is a given constant), subject to the boundary conditions $u(0) = u(1) = 0$. (Briggs, 1987).

One approach to solving this problem is to divide the domain of the problem, $\{\ x: 0 \leq x \leq 1\ \}$ into N subintervals, each of width $h = 1/N$, by creating $N+1$ gridpoints $x_j = jh$, $j = 0,...,N$. This is the finest grid, which is referred to as Ω^h. Grids in which the spacing between gridpoints is increased are referred to as coarser grids, denoted by Ω^{2h}, Ω^{4h}, etc. The coarsest grid will be denoted $\Omega^{\tilde{h}h}$. In order to reduce notational complexity, two conventions will be introduced. First, the discrete vector $v(x)$ is used to replace the continuous function $u(x)$ as a reminder that the problem has been discretized. Secondly, v_j is used as shorthand for $v(x_j) = v(jh)$, and f_j is used as shorthand for $f(x_j)$. By replacing the second derivative in the original problem by a second order finite difference approximation, the differential equation is approximated by the second order difference equations $\frac{-v_{j-1}+2v_j-v_{j+1}}{h^2}+\sigma v_j=f_j$, $v_0 = v_N = 0$, for $j=1,2,...N-1$.

Since the above difference equation must be satisfied at each interior grid point, this discretization results in a system of N-1 equations and N-1 unknowns. In matrix form, this system can be expressed as

$$\frac{1}{h^2}\begin{bmatrix} 2+\sigma & -1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ -1 & 2+\sigma & -1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & -1 & 2+\sigma & -1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & -1 & 2+\sigma & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 2+\sigma & -1 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 & 2+\sigma \end{bmatrix} \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-3} \\ v_{n-2} \\ v_{n-1} \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-3} \\ f_{n-2} \\ f_{n-1} \end{bmatrix}.$$

Calling the above matrix **A**, this can also be expressed as $(1/h^2)\boldsymbol{A}\ \boldsymbol{v} = \boldsymbol{f}$. The matrix A is tridiagonal, symmetric, and positive definite, of dimension *(N-1)* by *(N-1)*.

From a given current approximation, a Jacobi iteration for solving this system of equations solves the *i*th equation for the *i*th unknown. For example, if $\mathbf{v}^{(old)}$ is the current approximation to the solution, and $\mathbf{v}^{(new)}$ is the updated approximation, a Jacobi iteration step would give $v_i^{(new)} = \frac{1}{2+\sigma}\left(h^2 f_i + v_{i-1}^{(old)} + v_{i+1}^{(old)}\right)$, and a Jacobi sweep consists of one pass through all *N-1* equations. The generic term for an iterative sweep such as this is a *relaxation.*

To gain some insight into how the Jacobi method works, and to see what benefit is gained from a multigrid approach, assume for now that $\boldsymbol{f(x)} = \boldsymbol{0}$ and $\sigma = 0$. That is, solve the homogeneous differential equation $\boldsymbol{u_{xx}} = \boldsymbol{0}$. The advantage of this is that the exact solution, $\boldsymbol{u(x)} \equiv \boldsymbol{0}$, is known, and the error in any approximate solution $\boldsymbol{v}$ is simply -

v. In particular, the size of the initial guess is the same as the initial error, and we can watch how it evolves.

Consider an initial approximation consisting of a single Fourier mode. That is, v is a vector whose j^{th} component is $\sin\frac{jk\pi}{N}$, $j=1\ldots N-1$, where k is the wave number of the mode. Applying a Jacobi iteration to it produces an interesting result. If the mode is in the high end of the spectrum (i.e., $k \geq N/2$), then the error is reduced quickly. If the mode is in the low end of the spectrum, very little error reduction is achieved.

If an initial solution consisting of a combination of Fourier modes is used, after only a few iterations the high frequency modes of the error are almost completely removed, while the low frequency modes are relatively untouched. The result is that the error is 'smooth', consisting only of low frequency sine waves. Faced with this smooth error, a local operator like a finite difference approximation is unable to advance quickly towards the exact solution.

Multigrid algorithms attack this problem by transferring the problem from a grid where the smooth nodes are low frequency to one where they will appear to be higher frequency. If the grid spacing on an interval is doubled, then the number of modes recognizable on that interval is halved, so a Fourier mode which is in the low end of the spectrum when there are $N-1$ possible modes, will move towards the higher end of the spectrum when the possible values only go up to $N/2$. This is demonstrated graphically in Figure 2, where the k=16 mode appears to be higher frequency on the coarse grid (cycling every four points) than on the fine grid (where it cycles every eight points) (Briggs, 1987).

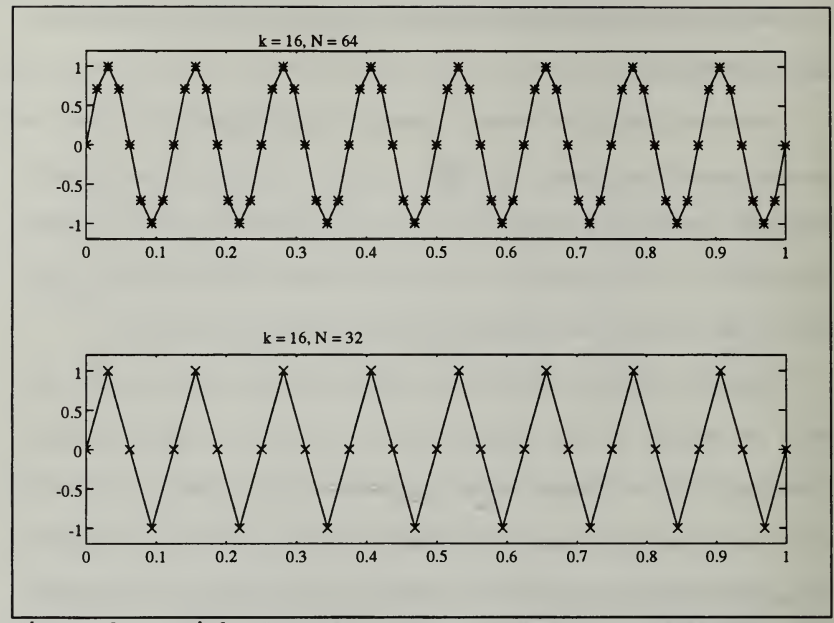


Figure 2. High Frequency Wave Appears Low Frequency on Coarser Grid

B. COARSE GRID CORRECTION AND THE MULTIGRID V-CYCLE

The following notational conventions will be used to distinguish between different iterations and levels during the multilevel process. In the case of discrete variables, functions and matrices, superscripts are used to indicate which multigrid level they apply to. For example, if the grid spacing on the finest level is h, then variables on this level would have the superscript h. The next coarser level will have the superscript $2h$. (This assumes a 2:1 ratio of the grid spacing. Subscripts of scalar quantities denote vector

index, as in v_j. Vectors and matrices will be denoted by bold print. Variables and functions with no superscripts or subscripts are continuous.

During the course of the multilevel algorithm, the problem is addressed on various scales, as indicated in the previous section. In doing this, many parameters of the problem, including the current estimate of the solution, the error, the right hand side vector, and the finite difference operator A are transferred from fine grids to coarse grids and back again. The act of transferring to a coarse grid from a fine grid is called *restriction.* Transferring to a fine grid from a coarse grid is known as *prolongation*, or *interpolation.* The letter I indicates either an interpolation or restriction operator. For example, $\mathbf{e}^h = I_{2h}^h \mathbf{e}^{2h}$ indicates that e^h is the interpolation (from Ω^{2h} to Ω^h) of the vector e^{2h}. Likewise, $\mathbf{e}^{2h} = I_h^{2h} \mathbf{e}^h$ is the restriction of the vector e^h from Ω^h to Ω^{2h}. Notice that the subscript of the operator and the superscript of the operand must agree, as do the superscript of the operator and the superscript of the result (Brandt, 1984).

In general, any error vector may be composed of error components at many frequencies, from very high (oscillatory) to very low (smooth). The multigrid approach is to treat each component of the error on a grid where it appears to be high frequency. For the partial differential equations which multigrid was initially designed to solve, the components of the vector x^0 are neighboring elements of a discretized function. This discretization could be thought of as sampling the continuous function in some way, and either taking the value of the function at a point, or accumulating the value over a specific region. On successively coarser and coarser grids, the distance between neighboring

sample points is increased, and the function values at the grid points carry information for larger portions of the original domain.

The error of any approximation can be decomposed into a finite sum of Fourier modes on a finite grid. Only modes with a wavelength between $2h$ and $2Nh$ can be represented on a grid with $N+1$ gridpoints (including the endpoints) and a grid spacing of h, and these modes are the Fourier modes with wavenumbers 1 through N. The highest frequency component oscillates over the wavelength $2h$. Sine waves of higher frequency would appear as lower frequencies through a process called *aliasing*. A simple demonstration of this process is observed in movies, when a rapidly spinning wagon wheel appears to be spinning slowly, or backwards, because the discrete sampling frequency of the camera is slower than the actual frequency of the wheel. Similarly, the lowest frequency component which can be represented is that which has a wavelength of twice the interval length, that is, the one which completes half a cycle in the number of sample points in the interval.

A Fourier mode on the fine grid, when transferred to a coarser grid, will have the same wave number as on the fine grid (Figure 2). However, since only half as many wave numbers can be represented on the coarse grid, this same mode will be on the high end of the spectrum on the coarse grid. That is, the error which was smooth on the fine grid appears to be high frequency on the coarse grid. The iterative method applied on this coarser grid will reduce the remaining error as if it were actually the high frequency component.

The concept of transferring the problem to a grid on which the smooth error will appear to be high frequency, further reducing the error by iteration (relaxation), and then interpolating a correction back to the original level, is known as the *coarse grid correction scheme*. This algorithm for finding a solution to $Ax=b$ can be expressed as follows:

1. Starting with an approximate solution x_0, run a few iterations on the fine grid, to produce the approximation x^h. This process is called 'relaxation'. After very few iterations, the error is smooth.

2. From the above relaxation, compute the residual, $r^h = b^h - Ax^h$.

3. Transfer r^h to the next coarser grid, $r^{2h} = I_h^{2h} r^h$.

4. Solve $A^{2h} e^{2h} = r^{2h}$ on the coarse grid. Because this problem is on the coarse grid, with fewer data points, this operation requires fewer arithmetic operations than on the fine grid.

5. Interpolate e^{2h} back to the fine grid as an estimate for e^h and correct the estimate by $x^h \leftarrow x^h + I_{2h}^h e^{2h}$. Since e^h is smooth as a result of local relaxation, then e^{2h} is a good estimate for it. (Briggs, 1987)

Of course, this algorithm begs the question of how to solve $A^{2h} e^{2h} = r^{2h}$ on the coarse grid. The answer is that the procedure can be applied recursively, until ultimately a coarse enough grid is reached where the solution is readily available. On the coarsest grid possible, the problem is simply a single equation and a single unknown, and solving this requires only a single floating point operation. The process of relaxation, restriction, relaxation, etc., until the coarsest grid is reached, followed by interpolation, correction,

relaxation, interpolation, etc., until returning to the finest (original) grid, is known as a V-cycle, and is the simplest of multigrid schemes (see Figure 3).

Three important points should be made about the coarse grid correction V-cycle. The first is that, at all but the finest grid level, the scheme works on the residual equation, not the original equation. The reason that this is important is that, after a few relaxations of an iterative solver, the error will be fairly smooth, and therefore may be accurately represented on the coarse grid. No such claim can be made about the approximate solution to the original equation (Brandt, 1984).

The second point is that at every level of the coarsening process, relaxation steps can occur. The result of this is that the various components of the error are being attacked on the most appropriate level, and that, by the time the coarsest level is reached, all components of the error have been reduced.

Finally, although it may appear at first that the structure of the V-cycle is adding significantly to the work of the problem, it must be remembered that at each deeper level of the V, the problem being solved is getting smaller and smaller. The result of this is that, typically, the total work done in a single V-cycle is on the order of two to three times the work of a single relaxation sweep of the iterative solver, and the results are greatly improved.

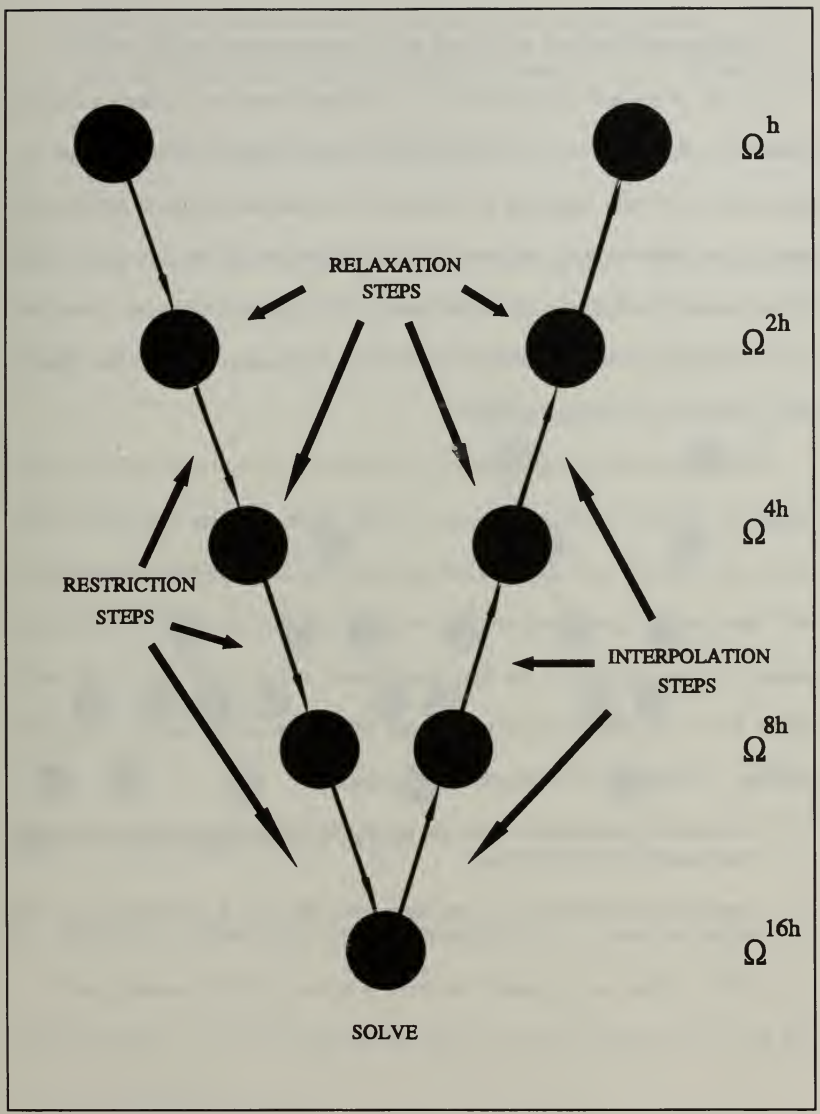


Figure 3. Multigrid V-Cycle

C. NESTED ITERATION AND THE FULL MULTIGRID (FMG) CYCLE

The second element which, together with coarse grid correction, constitute multigrid is known as *nested iteration.* The idea behind nested iteration is that the better the approximation is at the beginning of a V-cycle, the better the end result will be. To obtain a good initial solution, one can first solve the problem on the next coarser level and interpolate the solution to the original level. This approach is efficient in that the cost of computing an initial solution is small, since the problem on the coarse level is much smaller than the original problem.

In the *Full Multigrid (FMG) Scheme*, the problem is solved on the coarse level by performing a V-cycle starting on that level. V-cycle implies returning step by step to the coarsest grid, and then back up to the current level. This idea is applied recursively, so that in order to get a good initial estimate for this coarse level the problem is solved by running a V-cycle starting on the next coarser level, and so forth down to the very coarsest level. The (FMG) scheme incorporates both nested iteration and the coarse grid correction. A statement of the algorithm is as follows:

1. Start on the coarsest level. Since the problem on the coarsest level is very small, it can usually be solved directly.

2. Interpolate the solution to the next finer level and relax a few iterations on the original equation.

3. Perform a V-cycle from this level, working on the residual equation.

4. If on the original (finest) level, stop. Otherwise, return to step 2. (Brandt, 1984)

In other words, each V-cycle is preceded by a smaller V-cycle which starts at a coarser level, as shown in Figure 4. The purpose of relaxation after interpolating to a higher grid level is that, in most cases, interpolation will introduce a small high frequency error into the approximation. However, since relaxation methods eliminate high frequency error quickly, one or two iterations are sufficient to remove this error.

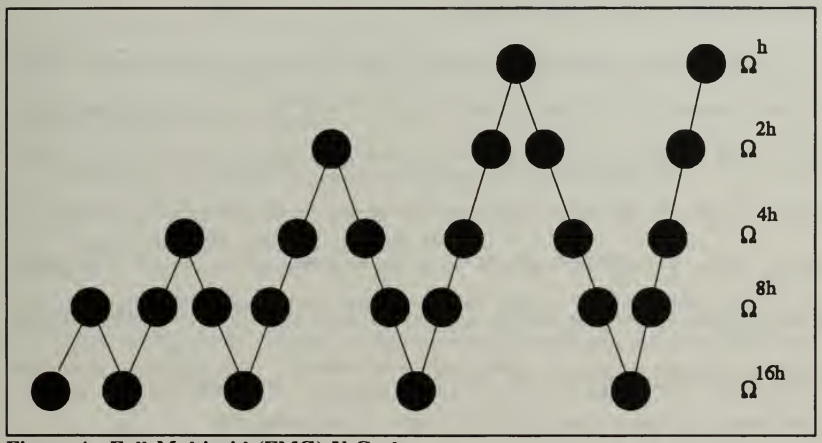


Figure 4. Full Multigrid (FMG) V-Cycle

D. THE FULL APPROXIMATION SCHEME (FAS)

The arguments for the coarse grid correction scheme do not carry over directly to nonlinear problems. If N is a nonlinear operator and N^h its' representation on the fine

grid, f the right hand side vector, u the exact solution and v the approximate solution to the equation $N^h u^h = f^h$, then the residual, r is defined:

$$r^h = f^h - N^h v^h$$
$$= N^h u^h - N^h v^h.$$

Note that in general, $N^h u^h - N^h v^h \neq N^h(u^h - v^h)$, since a nonlinear operator such as N is not distributive. However, $u^h - v^h = e^h$, so, in contrast to the linear case, $N^h u^h \neq r^h$. Note that the error e does not solve the same set of equations as the solution v when the right hand side is replaced by the residual.

For this reason, the *full approximation scheme* (FAS) was developed. Let N be the nonlinear operator in the equation $Nu=f$. Define $v^{2h} = I_h^{2h} v^h$ to be the restriction of the approximate solution v^h. The residual, $r^h = f^h - N^h v^h$, is the difference between the original right hand side, f^h, and the nonlinear operator acting on the approximate solution, v^h. Finally, define $r^{2h} = I_h^{2h}(f^h - N^h v^h)$ to be the restriction of the residual. The residual is the difference between the operator acting on the exact solution and the operator acting on the approximate solution, so $f^h - N^h v_k^h = N^h u^h - N^h v^h$. This equation is analogous to the residual equation used in the FMG cycle. Transferring this residual to the $2h$ grid gives $I_h^{2h}(f^h - N^h v^h) = N^{2h} u^{2h} - N^{2h} v^{2h}$. Using $v^{2h} = I_h^{2h} v^h$, this becomes $N^{2h} u^{2h} = I_h^{2h} f^h + N^{2h}(I_h^{2h} v^h) - I_h^{2h}(N^h v^h)$. Observe that this is the coarse grid version of $N^h u^h = f^h$, but that a correction term, namely $T_h^{2h} = N^{2h}(I_h^{2h} v^h) - I_h^{2h} N^h v^h$, is required due to the nonlinearity. Formally, solving for u^{2h} gives $u^{2h} = (N^{2h})^{-1}(I_h^{2h}(f^h - N^h v^h) + N^{2h} v^{2h})$. In other words, the coarse

solution is the inverse coarse grid operator acting on the sum of the restricted residual and the coarse grid operator acting on the coarse approximation.

However, u^{2h} is the full solution on the coarse grid, (hence the name, full approximation scheme), and we only desire the correction, $u^{2h}-v^{2h}$. To get the correction and update the approximate solution, we take

$$v^h \leftarrow I_{2h}^h (u^{2h} - I_h^{2h} v^h) .$$

The important differences between FAS and the previous multigrid schema is that $N(u^{2h})$ is approximating the *solution* on the next coarser grid, not the *error*. A statement of the FAS algorithm is as follows:

1. Relax on the fine grid to get an approximation to the solution, v^h.

2. Restrict the right hand side to obtain f^{2h} .

3. Restrict the approximate solution and apply the operator to it to find $N^{2h}(I_h^{2h} v^h)$. Apply the operator to the fine solution and restrict the result to find $I_h^{2h}(N^h v^h)$. Compute $T_h^{2h} = N^{2h}(I_h^{2h} v^h) - I_h^{2h}(N^h v^h)$ and add it to f^{2h} .

4. Solve $N^{2h} u^{2h} = f^{2h} + T_h^{2h}$ to find u^{2h}.

5. Subtract the restricted initial estimate v^{2h} from u^{2h}, and interpolate the result to the fine grid.

6. Add the correction to the original approximation, $v^h \leftarrow v^h + I_{2h}^h (u^{2h} - I_h^{2h} v^h)$. (Brandt, 1984)

To solve the problem in step 4, the above scheme is applied recursively. As with the coarse grid correction scheme of the linear case, FAS is used recursively until the coarsest level is reached and the problem is easy to solve. Nested iteration is also used to create an FAS-FMG scheme.

E. MULTIGRID APPROACH TO THE TRANSPORTATION PROBLEM

Kaminsky (1989) wrote a Master's thesis at the Weizmann Institute of Science in Rehovot, Israel, under the guidance of Prof. Achi Brandt. The premise of the thesis was that multilevel techniques, which had been developed in relation to partial differential equations, and which had been applied with success to a growing number of areas, might be able to provide an improved solution method for optimization problems in general, and specifically, to the long transportation problem.

An optimization algorithm which most closely resembles a multilevel algorithm is aggregation/disaggregation, in which nodes are aggregated in a logical way in order to reduce the size of the problem, and the solution to the smaller problem is disaggregated to provide either an initial estimate or a bound for the solution to the original problem (Zipkin, 1980). The algorithm Kaminsky developed differs from this method in that his work extends the algorithm from a two level approach to a multilevel approach. That is, aggregated nodes are aggregated again, and again, until a problem which is trivial to solve exactly is reached. In multilevel terminology, the aggregation process is referred to as *restriction*, and the disaggregation is called *interpolation*.

Kaminsky required that the nodes in the problem occupy a location in physical space, and that the cost of the arcs connecting supply nodes to demand nodes obey the properties of physical distance (i.e., comprise a metric): $c_{ij} \geq 0$ for all i,j; $c_{ij} = 0$ only if nodes i and j are co-located; and $c_{ij} \leq c_{ik} + c_{kj}$ for all i, j, k. Such a problem may be termed a *geometric* long transportation problem. With this requirement in place, demand nodes were restricted (aggregated) based on their physical closeness, so that individual demand points were combined into regional demand centers, and so forth.

Kaminsky's restriction of the problem to those which had a physical relationship seems to be overly restrictive. Many problems that can be formulated as transportation problems have absolutely no physical space interpretation. For example, the assignment problem is a specialization of the transportation problem in which the cost of an arc is unrelated to physical separation of the connected nodes. In this problem, as well as in many other manpower or resource allocation problems, physical distance is meaningless. For this reason, a methodology for relaxing this modeling assumption is developed, as detailed in the next chapter.

IV. DESIGN AND IMPLEMENTATION OF THE V-CYCLE

There were three objectives in the first phase of research for the non-geometric long transportation problem. The first was to develop the control and data structures for implementing a multilevel V-cycle, i.e., design and program routines for restriction, interpolation, and local relaxation.

The second objective was to implement a working interface with GNET, which would be used as a 'black box' solver in the local relaxation phase of the V-cycle scheme. A subroutine version of GNET (Bradley, Brown and Graves, 1977) was provided by Professor Gordon Bradley, Naval Postgraduate School, to be used for this purpose.

The third objective was to perform initial testing of the V-cycle. This testing was to be very exploratory in nature. Its purpose was to point out ways the algorithm could be improved.

The first two objectives were met in the first preliminary design. Shortcomings in this design, as identified while pursuing the third objective, were addressed in the second phase of the design process.

A. FIRST PRELIMINARY DESIGN PHASE

Prior to addressing the questions of how the basic multigrid operations of restriction, interpolation, and local relaxation would be implemented, some general design decisions

had to be answered. First of all, Kaminsky (1989) required that the demand nodes occupy a physical location in space, and that a relationship exist between transportation costs and distances. In order to overcome this limitation, a change of coordinate axes from physical space to cost space is performed. For the $M \times N$ problem, cost space is the M-dimensional space in which each of the coordinate axes is the cost of shipping from one of M supply nodes. Each of the N demand nodes is placed in cost space at the point whose coordinates are the unit costs of shipping from the supply nodes to it. An example, for a 2×16 problem, showing the relationship between physical space and cost space is given in Table I, and depicted graphically in Figure 5.

Figure 5. Cost Space vs. Physical Space

Kaminsky's requirement that a relationship exist between cost and distance is connected to his method of restriction. He aggregates demand nodes based on physical

Table I. COST SPACE VS. PHYSICAL SPACE

DEMAND NODE	PHYSICAL SPACE X	Y	COST SPACE S1	S2
2	12	18	8	18
2	48	36	12	15
3	24	44	15	18
4	48	52	16	18
2	48	68	8	12
2	7	77	9	14
7	18	85	13	12
8	32	96	18	14
2	85	12	17	15
11	68	85	18	15
11	68	47	17	10
12	72	48	16	8
13	54	68	18	10
14	91	96	20	5
15	95	88	22	4
16	75	91	18	8

closeness, which makes sense if demand nodes physically close to each other have similar costs. Changing coordinate systems means that demand nodes which are close in cost space are aggregated directly, instead of aggregating nodes which are close in physical space and requiring a connection between distance and cost (see Figure 7 in the next section). The objective in both cases is to combine demand nodes which have similar costs of shipping from each supply node. By transferring the problem to cost space, a more direct path to this objective can be followed.

Using a cost-space implementation, the dimensionality of the problem equals the number of supply nodes. One way to lessen the work of any solution method somewhat is to transform from the original cost-space to a 'reduced dimension' space, and this is accomplished by subtracting the cost of shipping from one of the supply nodes (which must be connected to each demand node) from the cost of shipping from each supply node. The result is that, for one supply node, all the demand nodes map to the origin in cost space. We can show that while the objective function value is different for the new problem, a solution for one is equivalent to a solution for the other.

Let the long transportation problem be represented by a bipartite graph G, with M supply nodes and N demand nodes, and suppose each supply node is connected to every demand nodes. Let K be the set of arcs in the problem. Let b be the $(M+N)$ length column vector whose first M entries are the supplies at the supply nodes and whose remaining N entries are the negative of the demands at the demand nodes, and let A be the adjacency matrix of the graph G; that is, each row of A is associated with a node in G, and each column of A represents an arc of G, such that for every arc k from supply node i to demand node j, column k of A has a +1 in row i and a -1 in row $M+j$. Let c be the vector whose k *th* element, $k=i+M(j-1)$, is the cost c_{ij} of shipping from node i to node j along arc k. If l is the index of a supply node which is connected to every demand node, define $\tilde{c}$ to be the vector whose k *th* entry is $\tilde{c}_k = c_{ij}-c_{lj}$. Then we can prove the following theorem.

Theorem 1: *For the M×N transportation problem, where* A, b, c, *and* $\tilde{c}$ *are defined as above,* x^* *is a solution to the problem*

$$\begin{aligned} &\textit{Minimize } z = c^T x \\ &\textit{Subject to: } Ax = b \\ &\qquad x \geq 0 \end{aligned} \qquad \textit{Problem 1}$$

if and only if it is a solution to the problem:

Proof:

$$\begin{aligned} &\textit{Minimize } z = \tilde{c}^T x \\ &\textit{Subject to: } Ax = b \\ &\qquad x \geq 0. \end{aligned} \qquad \textit{Problem 2}$$

$$\begin{aligned} \tilde{c}^T x &= \Sigma\Sigma(\tilde{c}_{ij} x_{ij}) \\ &= \Sigma\Sigma(c_{ij} - c_{ij}) x_{ij} \\ &= \Sigma\Sigma(c_{ij} x_{ij} - c_{ij} x_{ij}) \\ &= \Sigma\Sigma c_{ij} x_{ij} - \Sigma\Sigma c_{ij} x_{ij} \\ &= c^T x - c_{ij} \Sigma\Sigma x_{ij} \end{aligned}$$

But $\Sigma\Sigma x_{ij}$ is the total flow on all arcs, which, in the transportation problem is equal to the total supply, $\Sigma_{i=1}^{M} b_i$. Therefore,

$$\tilde{c}^T x = c^T x - c_l \Sigma_{i=1}^{M} b_i.$$

But $c_l \Sigma_{i=1}^{M} b_i$ is a constant, independent of x. Therefore, x^* is a solution to Problem 1 if and only if it is a solution to Problem 2. ■

The transformation of the costs to the reduced dimension space is linear in the number of arcs, and will allow for one fewer coordinate axis to be considered during the restriction process, since the transformed cost of shipping from supply node *l* to every demand node maps to the origin in cost-space. Most notably, the 'reduced-dimension' problem requires sorting the nodes (an order *M log M* operation which will be required during the restriction process) fewer times. Figure 6 is a depiction of the demand nodes

in the 2 supply node problem of Table I mapped to the one dimensional transformed cost space. Once the transformation to 'reduced-dimension' cost-space has been performed, the resulting problem may be solved with no further consideration of the transformation. Therefore, in the remainder of this work it is assumed that when an $M \times N$ problem is to be solved, it may be the reduced dimension version of a problem that was originally $(M+1) \times N$.

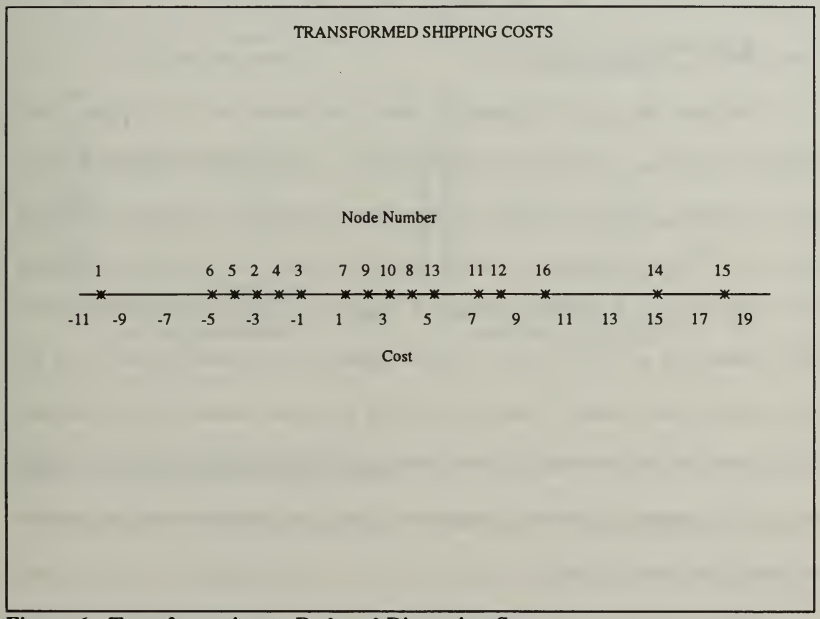


Figure 6. Transformation to Reduced Dimension Space

A general multigrid V-cycle pattern of restrict -- relax -- interpolate was implemented in the first preliminary design. The first step of the design was to develop algorithms for each of these operations. Each operation will be addressed separately.

B. RESTRICTION

When aggregating demand nodes, three attributes need to be defined on the coarse grid: the demands, flows and costs. One scheme seems natural for demands and flows; namely, the "trivial" restriction we describe next. Demands of the nodes being aggregated are added together, and flows from each supply node to the demand nodes are added together. That is, if nodes j and l are being aggregated, $I_k^{k+1}(D_j,D_l) = D_j + D_l$. For each supply node i let ij be the arc from node i to node j, and let il be the arc from node i to node l, then $I_k^{k+1}(x_{ij},x_{il}) = x_{ij} + x_{il}$.

Restricting the cost of shipment is more complicated, and no obvious 'best' approach is apparent, however several options exist. The simplest of these is, if nodes j and k are being aggregated, define the coarse cost to equal the minimum of the fine costs, i.e., $I_k^{k+1}(c_{ij},c_{il}) = \min(c_{ij},c_{il})$. Other simple schemes, such as using the maximum of the fine costs, or a weighted average $I_k^{k+1}(c_{ij},c_{il}) = \dfrac{\alpha_j c_{ij} + \alpha_l c_{il}}{\alpha_j + \alpha_l}$ seem equally valid. Some choices for α_j and α_l are 1 (equal weighting), the demands D_j and D_l, or the flows x_{ij} and x_{il}, from node i to nodes j and l from a previous solution. In this last case, provision must be made for the case where there is zero flow on both arcs. In the initial phase of the research, all of these schemes were coded, and experimentation and analysis performed using each of them.

Initial restriction of the problem was accomplished as follows. The demand nodes are first sorted by increasing cost of shipping from supply node 1, and divided into two groups about the median of the sorted cost (Figure 7a). All sorts in the algorithm are accomplished using a shell sort routine (Aho, et. al., 1982).

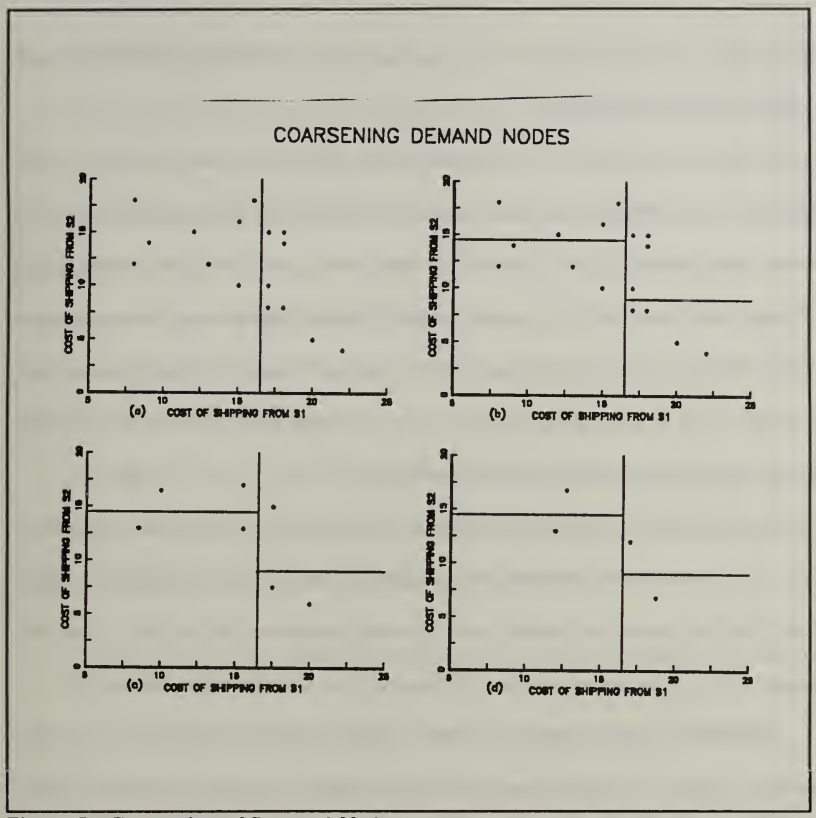

COARSENING DEMAND NODES
COST OF SHIPPING FROM S2
(a) COST OF SHIPPING FROM S1
(b) COST OF SHIPPING FROM S1
(c) COST OF SHIPPING FROM S1
(d) COST OF SHIPPING FROM S1

Figure 7. Coarsening of Demand Nodes

This procedure results in two groups of demand nodes. The first group are the less expensive nodes for shipping from S_1, and the second group are the more expensive. Each of these groups were next ordered by increasing cost of shipping from S_2. Dividing each group in half about its median results in one group which is expensive for both supply nodes, one group which is inexpensive for both nodes, one group which is

expensive for S_2 and inexpensive for S_1, and one group which is expensive for S_1 and inexpensive for S_2 (Figure 7b).

If there are more than two supply nodes in the problem, the above process is continued. The four groups are each sorted by cost of shipping from supply node 3, then divided into smaller groups if necessary and sorted again, until the demand node subgroups have been sorted by cost of shipping from all supply nodes in the problem. The result is that, in the final ordering, any two consecutive nodes have similar costs from all supply nodes as long as no 'boundaries' between node groups are crossed. That is, each group is disjoint, and ordered independently.

Once the node ordering was established, that process never had to be repeated. Since all of the restriction methods used to aggregate costs involved a weighted average of the fine level costs, the ordering was preserved throughout the V-cycle. Thus, the adjacent nodes in the above ordering are aggregated as shown in Figure 7c and 7d.

Thus, if Ω^1 is the original grid, then restriction to Ω^2, the second level of the problem, is achieved by adding the demands and averaging the costs of neighboring pairs of demand nodes in the node ordering list. To implement this, the coarse costs and demands are indexed by the first node of the pair. To restrict to Ω^k, the k th level, demands and costs which are distance 2^k apart in the node list are combined, where k is the level in the V-cycle, numbered from 0 (at the finest level) to $\tilde{k}$ at the coarsest level, $\Omega^{\tilde{k}}$.

C. INTERPOLATION

Eventually, all the demands and costs are restricted until the coarsest grid, Ω^k, is reached, where there is only a single coarse demand node. This node necessarily receives all of the flow from all of the supply nodes, and, being the only possible flow assignment on the coarsest grid, this is the optimal solution. This solution is then interpolated to the next higher level. Interpolation here means to solve an $M \times 2$ transportation problem, where M is the number of supply nodes, and the two demand nodes are the component nodes of the single Ω^k demand node.

A heuristic which is a special case of Vogel's approximation method (Bazaraa et. al, 1990) is used to solve this $M \times 2$ problem. In this special case ($M=2$), the solution obtained by this method is optimal. The method proceeds as follows:

1. For each supply node, determine the difference in cost of shipping to the two fine grid demand nodes.
2. Rank the M supply nodes in order of these differences, largest to smallest.
3. Allocate flow starting to the least expensive arc from the supply node with the greatest difference. In the event of equal differences, the arc with the smallest cost is chosen.
4. If the supply at the current node is exhausted, remove that node from the problem.
5. If the demand at the current demand node is completely satisfied, remove that node from the problem, allocate the remaining available supply from the current supply node to the second demand node, and remove the current supply node from the problem.
6. If there are any remaining nodes in the problem, return to step 3.

As an example of this procedure, consider the five by two problem shown in Figure 8. The five supply nodes S_1, S_2, ...S_5 have, respectively, 15, 12, 16, 18 and 1 units of the commodity to deliver. The demands of the two demand nodes D_1 and D_2 are 30 and 45. Let $\boldsymbol{d} = (4\ 8\ 4\ 6\ 1)^T$ be the vector whose i th entry is the difference between shipping cost from supply node i to the two demand nodes (the costs themselves are given for each arc in the figure).

Sorting from largest to smallest value of d_i, the supply nodes are ordered (2,4,1,3,5). Note that, while the differences for nodes S_1 and S_3 are the same, the arc from node S_1 to node D_2 is less expensive than either of the arcs incident from node S_3. Starting with node S_2, then, as much flow as possible is sent along the least expensive arc. In this case, that is the arc to demand node D_1. Since this demand exceeds the available supply from node S_2, all of the flow from node S_2 goes along this arc. Similarly, node S_4 and then node S_1 send all of their supply to node D_2.

When node S_3 has sent 12 units of flow along its least expensive arc, the demand at node D_2 is completely met. Thus node S_3 sends its remaining units to node D_1, as does node S_5. Although the arc from node S_5 to D_2 is less expensive, the demand at D_2 has been met from supply nodes where the difference in arc costs is greater.

We can show now that because of the special structure of the $\boldsymbol{M}\times\boldsymbol{2}$ problem, i.e., the fact that there are only two demand nodes, this procedure produces an optimal solution.

The algorithm may be described as follows. Let a transportation problem with M supply nodes and 2 demand nodes be represented on a bipartite graph $\boldsymbol{G}$ with $K=M+1$

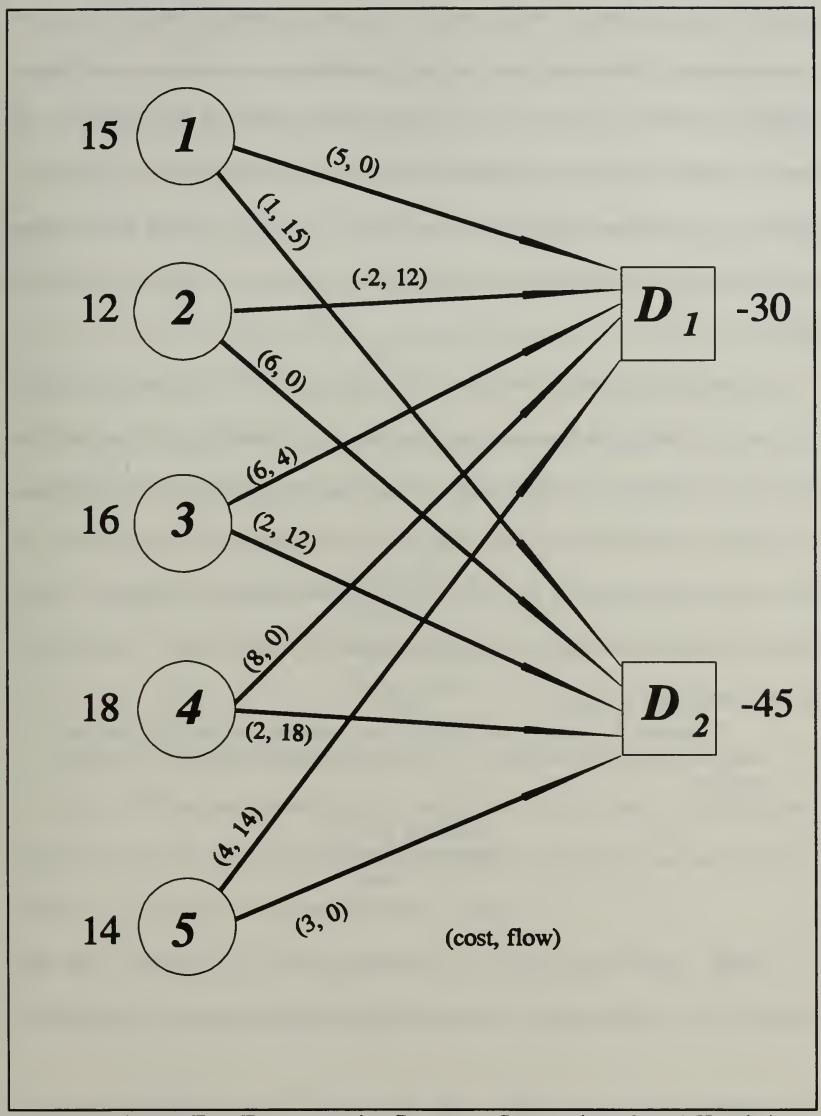


Figure 8. Five by Two Transportation Problem: Illustration of *M*×2 Heuristic

arcs. Let each supply node i be connected to both demand nodes by arcs of cost c_{i1} and c_{i2}, respectively. Let $\boldsymbol{b}$ be the vector whose first M elements are the amount of supply available at nodes S_1 through S_M and whose last two entries are the negative of the demand at nodes D_1 and D_2, respectively. Let $\boldsymbol{x}$ be the vector whose $i+M(j-1)$ st element, denoted $x_{(ij)}$, is the flow along arc (i,j) from node S_i to node D_j, and let $\boldsymbol{c}$ be the vector whose $i+M(j-1)^{st}$ element is the cost of arc (i,j). For every arc (i,j), let $\boldsymbol{x_{ij}}$ be determined as follows:

For each supply node S_i, determine $\boldsymbol{d_i=|c_{i1}-c_{i2}|}$ for each supply node i. Put the d_i's in order, breaking ties by comparing the costs of the original arcs. Starting with the node k whose difference is largest, assign as much flow as possible to the less expensive of c_{k1} and c_{k2}, until either the supply at that node is exhausted or the demand is met. If there is any remaining supply at node k, assign it to the remaining demand node. Next, repeat the process at the supply node whose difference is the next largest. Continue until all flows have been assigned.

Theorem 2: *Let $\boldsymbol{x}$ be the vector of flows assigned for the M×2 problem using the algorithm given above. Then $\boldsymbol{x}$ is the optimal solution to the M×2 problem*

$$\begin{aligned} &\textit{Minimize } z=\boldsymbol{c}^T\boldsymbol{x} \\ &\textit{Subject to: } \boldsymbol{Ax}=\boldsymbol{b}\,. \\ &\qquad\qquad \boldsymbol{x}\geq\boldsymbol{0} \end{aligned}$$

Proof: Assume that the flow, $\boldsymbol{x}$, as determined above is not optimal. Then there is a flow $\boldsymbol{x^*\neq x}$ which is optimal. Number the supply nodes so that they correspond with non-increasing values of d_i. Starting with supply node S_1, compare the flow determined

the node with the first difference between x^* and x node i, and let arc $(i,1)$ be the arc between node i and demand node D_1, and arc $(i,2)$ the arc between i and demand node D_2. Without loss of generality, assume $c_{i1} \leq c_{i2}$. Let $\Delta = x_{i1} - x_{i2}^*$. Since x_{i1} is as large as it can be (by construction), then x_{i1}^* must be less than x_{i1}, so $\Delta > 0$. That is, to reach optimality the flow from node i to demand node D_1 must be decreased by a positive amount. Likewise, the flow from node i to demand node D_2 must be increased by Δ so that the total flow out of supply node i remains unchanged. The total flow from all supply nodes after node i into demand node D_1 must be increased by Δ, and the total flow from subsequent supply nodes into demand node D_2 must be decreased by Δ so that the total demand for each demand node is met. Symbolically:

$$
\begin{gathered}
x_{i1} - x_{i1}^* = \Delta \\
x_{i2} - x_{i2}^* = -\Delta \\
\Sigma_{j>i}(x_{j2} - x_{j2}^*) = -\Delta \\
\Sigma_{j>i}(x_{j1} - x_{j1}^*) = \Delta \\
\Delta > 0
\end{gathered}
$$

Let $z^* = c^T x^*$ be the optimal objective function value, and let $z = c^T x$ be the objective function value for the flows computed initially. Then z^* is equal to z plus the difference in flows on each arc times the cost of that arc. That is,

$$z^* = z + (x_{i1} - x_{i1}^*)c_{i1} + (x_{i2} - x_{i2}^*)c_{i2} + \sum_{j=i+1}^{K}[(x_{i1} - x_{i1}^*)c_{i1} + (x_{i2} - x_{i2}^*)c_{i2}]$$
$$= z + \Delta c_{i1} - \Delta c_{i2} + \sum_{j=i+1}^{K}[(x_{i1} - x_{i1}^*)c_{i1} + (x_{i2} - x_{i2}^*)c_{i2}]$$

$$z^* = z + \Delta(c_{i1} - c_{i2}) + \sum_{j=i+1}^{k}[(x_{j1} - x_{j1}^*)c_{j1} - (x_{j1} - x_{s1}^*)c_{j2}]$$
$$= z + \Delta(c_{i1} - c_{i2}) + \sum_{j=i+1}^{k}[(x_{j1} - x_{j1}^*)(c_{j1} - c_{j2})]$$

But, by definition, $|c_{j1} - c_{j2}| = d_j$, so $c_{j1} - c_{j2} \geq -d_j$, and $-d_i \leq -d_j$, and as stated above,

$\sum_{j=i+1}^{K}(x_k - x_k^*) = -\Delta$, so

$$\sum_{j=i+1}^{K}[(x_{j1} - x_{j1}^*)(c_{j1} - c_{j2})] \geq \sum_{j=i+1}^{K}[(x_{j1} - x_{j1}^*)(-d_j)]$$
$$\geq (-d_i)\sum_{j=i+1}^{K}(x_{j1} - x_{j1}^*)$$
$$= -d_i(-\Delta)$$
$$= d_i\Delta$$

and so

$$z^* \geq \Delta(c_{i1} - c_{i2}) + d_i\Delta_i$$

and since $d_i = c_{i2} - c_{i1}$,

$$z^* \geq z + (-d_i)\Delta + (d_i)\Delta$$
$$z^* \geq z$$

Contradicting the assumption that $\boldsymbol{x}^* \neq \boldsymbol{x}$ is optimal. ■

D. NESTED ITERATION

During phase one, the goal of interpolation was to implement nested iteration. That is, to turn the optimal solution on a coarser level into a good initial solution on a finer level. This initial solution is identified to GNET, and GNET proceeds from this solution to optimality.

In order to identify the initial solution to GNET, the T (Tail) data structure is used to mark the arcs which are likely candidates to be in an optimal basis. Specifically, the sign bit of the T array entries for those arcs which have positive flow after interpolation is marked for identification.

GNET then proceeds in two phases. Starting with an artificial basis, the reduced cost of the arcs which have been identified by the multilevel algorithm are computed. If these arcs price out favorably, they are pivoted into the basis.

Up to a user-supplied maximum number of pivots, GNET then obtains the best flow possible using only these candidate arcs. Then, GNET starts its second solution phase. The remaining arcs are priced out, and, if favorable, pivoted into the basis. An optimal solution is found for the problem on the current level. This optimal solution is then immediately interpolated to the next higher level of the V-cycle, where the process is repeated.

Since the candidate arcs are still being priced out and pivoted into the basis, this approach proved very expensive in terms of computational effort. The lower limit on the number of pivots required in a V-cycle using this approach is $M\Sigma_{k=0}^{t}\frac{N}{2^k}$, where M is the

number of supply nodes, N the number of demand nodes on the finest level, and $\bar{k}$ the number of the coarsest level. As the number of levels increases, this lower limit increases to approximately $2MN$.

Although about half of these pivots are less expensive than the pivots on the finest level, it still is readily apparent that the amount of work being done would always be greater than the roughly $3N/2$ pivots being performed by GNET from a cold start on the finest grid. A less expensive approach had to be found, if the multilevel approach was to be useful.

The second approach to nested iteration was to eliminate the price out step of the previous method. That is, candidate arcs were identified to GNET and immediately pivoted into the basis, without examining the reduced cost of the arc. In order to implement this change, some new data structures had to be created and passed to GNET. These data structures identified the arcs which had positive flow after interpolation from the previous level. The structures were a HEAD array, a TAIL array, and a POS array, each of a length equal to the number of arcs with positive flow.

These new data structures were not in a reverse star arrangement, as opposed to the arcs which originally identified the network to GNET. These structures essentially constituted an arc list, in which an entry was required in each array for each arc. The HEAD array identified the demand node, the TAIL array identified the supply node, and the POS array indicated where in the original T array the arc was located. This facilitated GNET finding costs and flows. In addition, a count of the number of arcs with positive flow was also passed.

To test this new algorithm, the optimal solution on the finest level is passed in to GNET to see how quickly it would return it. Some additional pivots are performed, beyond those required to bring the optimal arcs into the basis. This indicates that, due to the order that the arcs are pivoted in, some of the arcs were being removed from the basis after being added to it, so that a little extra work is being performed in order to return to optimality.

However, the amount of additional work is very small, and starting with the optimal solution, GNET returns an optimal solution very quickly. This result is promising, since GNET was actually starting with an artificial basis and pivoting in all of the optimal arcs, yet the time required to reach optimality is only 20% of that required from a cold start. This would indicate that, for a starting solution sufficiently close to the optimal solution, a significant time savings could be realized.

When this new subroutine is called from the multilevel algorithm, however, the time savings as compared to a cold start at each level are very small. This means two things. The first is that, although on any given level running GNET from a cold start is slower than using GNET with the interpolated solution from the previous level, the overall running time of the algorithm using all levels is greater than a single call to GNET from a cold start on the finest level. The second is that the initial solution determined by interpolation of the coarser level optimum solution was not sufficiently close to the optimum solution on the fine level.

We believe that the main reason a multilevel approach failed to improve the performance of GNET is that the primal network simplex algorithm is not a local

operator. It expects and produces global information, and does so very efficiently. Since GNET is not characterized by the usual problem which multigrid fixes, i.e., a numerical process which stalls due to the inability to propagate local information quickly, little benefit is gained from using GNET in a multilevel approach.

Up to this point, the algorithm we had been using is really closer to nested iteration than to a genuine multilevel V-cycle. The restriction portion of the V only identifies the nodes to be aggregated, and combines demands and costs. No relaxation is being performed on the way down the V. In order to move closer to a standard multilevel algorithm, and to try to provide a better starting solution after interpolation, we now consider a full multigrid cycle algorithm.

V. THE FULL MULTIGRID ALGORITHM

In order for a full multigrid algorithm to show an improvement over the V-cycle algorithm, there must be a reason for revisiting the coarsest grid again and again. Repeated iterations on the same problem, using the same restriction method and the same interpolation method, would result in the same answer at a much higher computational cost. The ultimate goal is to reach an optimal solution at the finest level. All of the components of a multilevel algorithm need to work together to achieve that goal. In the simple V-cycle of the previous chapter, the optimal solutions to the subproblems on the coarser levels are not interpolated to the optimal solution on the finest level.

Recall from Chapter III that, in a full multigrid V-cycle, an initial approximation to the solution on any level is obtained by interpolation from the solution on the next coarser level, which is itself the result of an FMG cycle. During the downstroke of each V-cycle, the goal is to obtain a new problem on the coarsest level which will yield a good solution to the problem on that level, and ultimately lead to a good initial solution to the problem on the next finer level. There are three aspects of the algorithm, namely restriction, interpolation and local relaxation, any one or more of which might need to be changed so that optimality would be achieved.

Of these three procedures, we only experimented with restriction and local relaxation in the current research. In every case, the interpolation method described in Chapter IV was used to move from Ω^{k+1} to Ω^k. This was done for two reasons. First of

all, none of the other proposed schemes, such as a simple greedy algorithm, had the intuitive appeal of this scheme. The total complexity of this scheme is *O(N)* (Kaminsky, 1989), and an optimal solution is guaranteed as demonstrated in Chapter IV. Secondly, this was the method used by Kaminsky in his research, and he reported this as an "optimal disaggregation method" (Kaminsky, 1989).

This chapter will present the various restriction and local relaxation procedures investigated. The restriction methods studied can be divided into two categories. The first being variations on the weighted averaging of the costs, and the second being implementation of an approach which bears similarity to the full approximation scheme (FAS), through the use of dual multipliers. The local relaxation methods investigated can be divided into those utilizing local optimization, and one using a cycle detection and removal algorithm.

A. VARIATION OF COARSENING SCHEMES

1. Flow and Demand Weighting

In the single V-cycle algorithm, only a single method of restriction is used throughout the cycle. In the full multigrid cycle, the opportunity of using different coarsening methods is gained, since coarse level nodes are visited and revisited in each V-cycle. Demands and costs are initially coarsened with no knowledge of the ultimate flows on the arcs. Within each V-cycle, flows and costs are restricted with an initial solution at hand. There is no reason that the restriction method used within a V-cycle should be the same as the method used for initial coarsening. In fact, at the coarsest level

of each of the V-cycles in the FMG, the problem will always have a single demand node which requires the total of all the supply in the problem, and an arc from each supply node to this demand node carrying all of the available supply. The only possibility for improving the problem on the coarsest level, and thereby improving the result on the finest level, is by changing the costs of these arcs.

During the initial coarsening process, when the original problem is first transferred to the coarsest level so that the FMG cycle can begin, costs are restricted by either choosing the minimum cost of the two fine level arcs, or by computing a weighted average based on the demand of the nodes on the fine level, with demand-weighting producing a more nearly optimal solution. Within the V-cycles of the full multigrid algorithm, weighted averages based on either flow or demand were investigated, since, in the V-cycle algorithm, they gave better results than using the minimum or maximum cost. That is, the two methods of restricting costs which were implemented were demand-weighting,

$$c_{IJ}^{k+1} = I_k^{k+1}(c_{ij1}^k, c_{ij2}^k) = \frac{c_{ij1}^k d_{j1}^k + c_{ij2}^k d_{j2}^k}{d_{j1}^k + d_{j2}^k}$$

where J is the aggregation of fine nodes $j1$ and $j2$; and flow-weighting,

$$c_{IJ}^{k+1} = I_k^{k+1} c_{ij}^k = \frac{c_{ij1}^k x_{ij1}^k + c_{ij2}^k x_{ij2}^k}{x_{ij1}^k + x_{ij2}^k}.$$

Care must be taken in the case that flow on both arcs is zero. In this case, a simple average of the two costs is used.

2. Cost Conversion Using Dual Multipliers

In multigrid algorithms for solving linear partial differential equations, at all but the finest level of the problem the residual equation, $\boldsymbol{Ae=r}$, is solved, rather than the original equation, $\boldsymbol{Au=f}$. The reason for this is that the iterative methods used as local relaxation methods have the property that, after only a very few iterations the error in the current approximation is smooth, and this smooth error is well approximated by a transfer to the coarser grid. No such statement can be made about the solution to the original equation.

Reasoning by analogy, a multilevel approach to solving an optimization problem should also be using something similar to the residual equation. In a system of finite difference equations, the residual is a measure of by how much an approximate solution fails to satisfy the system of equations. In an optimization problem, there are two choices readily available which measure a similar quantity.

The first choice is $\boldsymbol{b-Ax}$, which is a measure of by how much the constraints of the problem are violated. While in a more general setting this approach might have advantages, all of the methods used in this work start from a feasible solution. That is, no constraints are violated and so $\boldsymbol{b-Ax}$ is always $\boldsymbol{0}$.

An alternative choice for a "residual" is the reduced cost of the non-basic arcs, since a necessary and sufficient condition for optimality is that the reduced cost for any non-basic arc, u_i-v_j-c_{ij}, be less than or equal to 0, where the u_i are the duals for the supply nodes and the v_j are the duals for the demand nodes. One interpretation of the dual multipliers is that of a 'node potential'. It is beneficial to increase the flow on an arc

only if the difference in node potential, u_i-v_j, is greater than the cost of the arc. (Bazaraa, et. al., 1990). This approach is the one used by Kaminsky (1989).

The duals are used in the following manner. Assume a solution has been interpolated to Ω^k from Ω^{k+1} by solving the $M\times2$ subproblems associated with each coarse-grid demand node. When restricting from Ω^k to Ω^{k+1}, the same $M\times2$ problems that are found in the interpolation are solved again. The duals of the supply nodes for this problem are then computed, and used as the costs of the arcs on Ω^{k+1}.

The computation of the duals is a simple process. In the non-degenerate case, a rooted spanning tree is created, rooted at an artificial node connected to the second demand node. The cost of the root arc is arbitrarily assigned a value of 0, and so the dual of the second demand node is also 0. From there, the dual of the 'turning' supply node, the one which has positive flow to both demand nodes, is computed. Since the reduced cost u_i-v_j-c_{ij}=0 for all basic arcs, and the dual of the second demand node, v_i, is 0, then $u_i = c_{i2}$, where i is the 'turning' supply node. For example, in the problem shown in Figure 9, the turning node is node S_2, and it's dual, u_2 is 3.

The dual of the first demand node is then u_i-c_{i1}. In Figure 9, this means that the dual of D_1, v_1=3-7=-4. Now that the dual of both demand nodes is known, the duals of all the supply nodes can be computed by adding the dual of the demand node which they supply to the cost of the arc connecting them. In the example, u_1=-4+(-4)=-8 and u_3=0+2=2.

In the degenerate case, there is no path from the second demand node to the first. In light of the restriction process, this means that a subset of the supply nodes

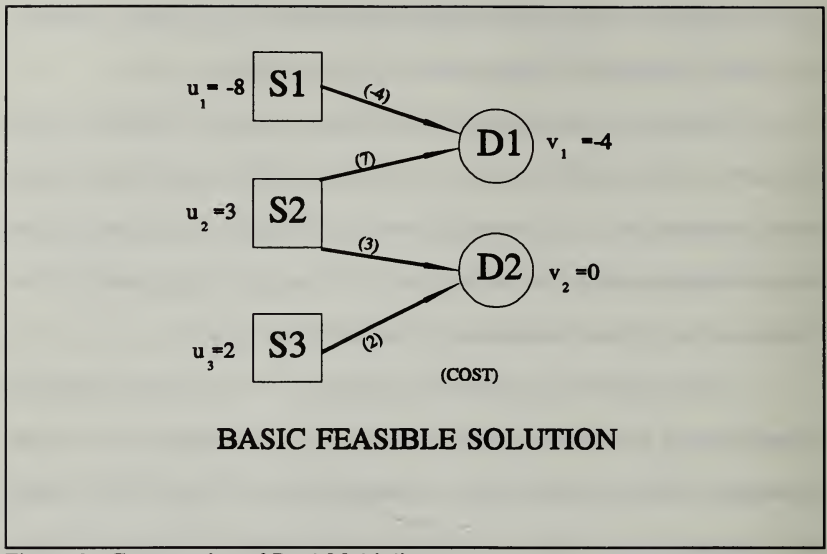


Figure 9. Computation of Dual Multipliers

feeding the coarse demand node supply one fine node, and the complement of that set feeds the other. An arc with zero flow could be chosen to enter the basis, and the duals computed as above based on the cost of this arc. Rather than use this somewhat arbitrary method, the costs of the arcs with positive flow on Ω^k are assigned to the corresponding arcs on Ω^{k+1}.

This approach yields marginally better results than those previously described. However, reasoning by analogy it appears that a step is missing in the algorithm. A multigrid algorithm for solving a partial differential equation uses the solution to the residual equation as a correction to the initial solution vector. That is, after solving the residual equation $A^k e^k = r^k$, the current approximation is corrected by $v^k \leftarrow v^k + e^k$. Such a

correction procedure is not yet implemented. Currently, during the interpolation process back to Ω^k, the costs from the original Ω^k are replaced in the problem, and once again the full problem (vice the residual equation) is solved. In order to completely follow the analogy of solving the residual equation in order to determine a correction to the initial solution, a scheme must be developed to extract the correction term from the solution on Ω^{k+1}, and add it to the initial solution. In short, an implementation of the Full Approximation Scheme (FAS) needs to be developed. Time constraints prohibited full consideration of this question for the present work, however it is the most promising avenue for further research.

B. LOCAL RELAXATION

1. Local Relaxation by Optimization

One of the keys to a properly functioning V-cycle is a good local relaxation method. The interpolation process, using the $M \times 2$ transportation problem algorithm, produces a locally optimal solution. That is, looking only at the flow to the fine level node pairs which comprise each coarse level node, the solutions produced by interpolation are optimal. However, when these demand nodes are considered as part of the global problem, the flow is not optimal.

The global problem is quite different from the union of several local problems. In each local problem, the reduced cost of each non-basic arc is less than or equal to zero since the current solution for it is optimal. However, the duals in the global problem are

considerably different, and subsequently the reduced cost $(u_i-v_j)-c_{ij}$ of a non-basic arc might be positive, indicating non-optimality.

Several techniques were investigated to move from the union of several locally optimal solutions towards a global optimum. By locally optimal, we refer to flows which are optimal on the subproblems created by the interpolation of a coarse demand node. By the nature of the interpolation procedure, once a flow was assigned to a coarse demand node, it could only then be divided amongst the nodes which comprised that demand node. This prevents any movement toward a globally optimal solution which does not have that particular total flow from each of the supply nodes to all of the demand nodes. A short example, illustrated in Figure 10, will help to demonstrate this problem.

Suppose that, in a three supply node problem, on level Ω^{k+1} demand node D_1^{k+1} is composed of fine level nodes D_1^k and D_2^k, and demand node D_2^{k+1} is composed of nodes D_3^k and D_4^k. Further, suppose that the demands at nodes D_1^k through D_4^k are 18, 5, 6, and 13 respectively. Total supply available from node S_1 is 15, from node S_2 is 15, and from node S_3 is 12. The fine level cost vector is

$$[c_{11}^k, c_{12}^k, c_{13}^k, c_{14}^k, c_{21}^k, c_{22}^k, c_{23}^k, c_{24}^k, c_{31}^k, c_{32}^k, c_{33}^k, c_{34}^k]^T$$
$$= [-10,\ 13,\ 2,\ 2,\ -2,\ 25,\ 20,\ 1,\ 8,\ 2,\ 7,\ 5]^T.$$

The coarse level demands would then be 23 at D_1^{k+1} and 19 at D_2^{k+1}, the costs would be

$$[c_{11}^{k+1}, c_{12}^{k+1}, c_{21}^{k+1}, c_{22}^{k+1}, c_{31}^{k+1}, c_{32}^{k+1}]^T = [-5,\ 7,\ 2,\ 3.895,\ 5,\ 5.632]^T,$$

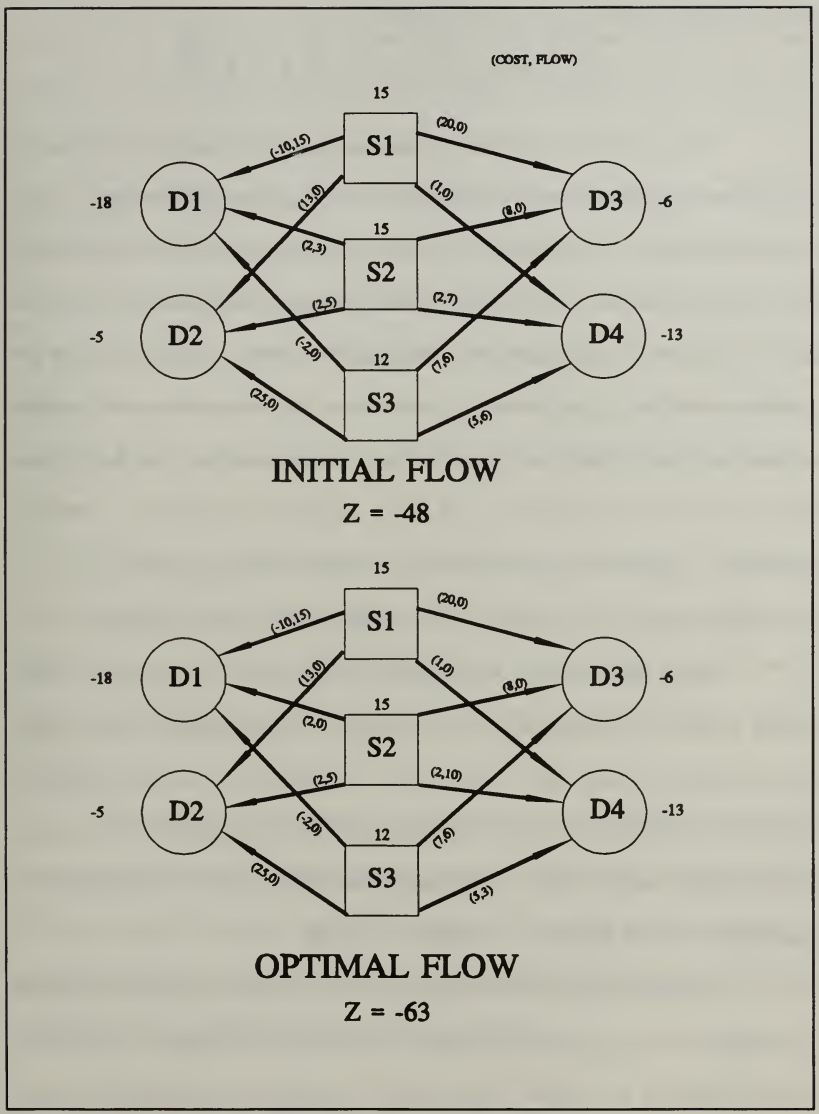


Figure 10. Comparison of Initial Flow and Optimal Flow

and the optimal solution to the coarse problem is

$$[x_{11}^{k+1}, x_{12}^{k+1}, x_{21}^{k+1}, x_{22}^{k+1}, x_{31}^{k+1}, x_{32}^{k+1}]^T = [15, 0, 8, 7, 0, 12]^T.$$

The result of the interpolation process is shown in the top half of Figure 10, with the two subproblems shown to the left and right, respectively, of the supply nodes. In each subproblem, one of the supply nodes is not supplying any of the demand nodes. This is caused by the fact that in an $M \times 2$ problem, at most one supply node can feed both demand nodes in the optimal solution. This means that on the coarse level, only one supply node will be supplying more than one of the two coarse demand nodes. The result is, when the coarse demand node is divided into its component fine nodes, at most two supply nodes are providing flow in each of the interpolation subproblems. Since, in Subproblem 1, supply node S_3 provides no flow to either demand node D_1^k or D_2^k, it cannot provide flow to any of the finer level nodes which comprise these nodes.

This is not as serious a shortcoming as it first appears to be. In fact, in a basic feasible solution to the global problem on any level, a maximum of M-1 demand nodes can be supplied by more than 1 supply node. The difficulty is not that node S_3 is prevented from supplying a portion of the nodes, but that the decision as to which nodes it may supply is made so early in the interpolation process, and, if relying solely on interpolation, that the decision is irreversible once made.

In Figure 10, the flow which results from the interpolation process is compared to the optimal flow for this problem, shown in the lower half of the figure. As discussed above, the decision not to allow supply node S_3 to provide flow to the coarse demand

node D_1^{k+1} was optimal in Ω^{k+1}, but results in no flow between S_3 and either D_1^k or D_2^k on Ω^k. Since the optimal solution includes flow on arc (3,1), it is impossible to reach optimality using interpolation alone.

In the PDE problem, the analogous difficulty is addressed by local relaxation. In our context this means crossing the boundaries imposed by the restriction process, so that flow may be shifted among the numerous locally optimal solutions, in such a way that the global objective function is improved. Several possible alternatives for local relaxation were studied, the first of which was designed to spread flow between demand nodes which had not been previously paired during the restriction and interpolation processes.

To begin this local relaxation, a small transportation problem is constructed using the second and third demand nodes on the current level from the ordered list of nodes (since the distribution of flows for the first and second nodes is already locally optimal). This problem is solved using the same $M \times 2$ algorithm which is used in interpolation.

Next a new $M \times 2$ problem is created using the (revised) flows to the third demand node and the flows to the fourth demand node. This method continues on until the last two nodes are reached. Recall from Chapter IV that nodes which are adjacent in the node ordering list have similar costs for shipping from all supply nodes, so comparing adjacent nodes is a reasonable approach to local relaxation.

The above method was altered to encompass a larger 'local' area in the relaxation step on each level. That is, in order to test whether an interaction of a greater number of demand nodes might improve the solution, the method was changed to accommodate a larger problem than $M \times 2$. In the new approach, a larger problem was selectable as the local relaxation problem. The problem size can be chosen to be $M \times 4$, $M \times 6$, and so forth up to $M \times 16$. GNET is used to solve each of these local problems, starting with the current flows as an initial approximation. Also, the user is allowed to specify how much overlap is desired between subproblems. For example, if a subproblem size of 4 with an overlap of 1 is selected, the first subproblem would consist of demand nodes D_1 through D_4, the second would consist of nodes D_4 through D_7, etc., as shown in Figure 11.

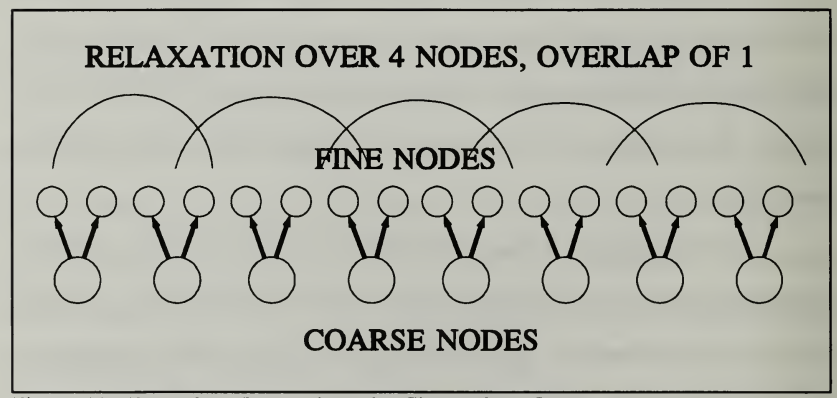


Figure 11. Four Node Relaxation with Single Node Overlap

These combinations were tested on a network consisting of five supply nodes and 1024 demand nodes, created by NETGEN (Klingman, et. al., 1974). Each supply node is connected to every demand node. The results are summarized in Table II. As is apparent from this table, little or no benefit is gained by increasing the size of the local area. Although in one case ($M \times 16$, overlap 0) the speed of the algorithm is marginally increased, the accuracy is the same for all combinations.

Table II. COMBINATIONS OF SUBPROBLEM SIZE AND OVERLAP

Combination	Running Time (s)	% over Optimality
M x 2	.269451	58.37
M x 4, overlap 0	.416265	58.37
M x 4, overlap 2	.625616	58.37
M x 8, overlap 0	.324426	58.37
M x 8, overlap 2	.433421	58.37
M x 8, overlap 4	.386384	58.37
M x 16, overlap 0	.255700	58.37
M x 16, overlap 2	.339964	58.37
M x 16, overlap 4	.299668	58.37
M x 16, overlap 8	.397563	58.37

2. Local Relaxation by Cycle Removal

The idea behind the methods of local relaxation discussed so far has been that, while the solution to each of the subproblems is locally optimal, the global solution is not. Therefore, information which is unavailable to a subproblem must be shared with it to move towards a globally optimal solution. The next technique applies a slightly different reasoning. The union of a pair of the locally optimized subproblems may be flawed, in that too many arcs might have flow on them, creating cycles and a less than optimal solution. This local relaxation technique attacks this feature of the interpolation process.

When interpolating from Ω^{k+1} to Ω^{k}, each coarse demand node generates two fine demand nodes and $M+1$ arcs with positive flow, in the non-degenerate case. If the subproblem is degenerate, then only M arcs will have positive flow. If there are $N/2$ demand nodes on Ω^{k+1}, the initial feasible solution to the fine level problem will have positive flow on between $NM/2$ and $(NM+N)/2$ arcs, depending on how many subproblems are degenerate.

The fine level problem has M supply nodes and N demand nodes, so an extreme point solution will have $M+N-1$ arcs with positive flow, absent degeneracy (Wolsey and Nemhauser, 1988). In the extreme degenerate case each supply node provides flow to a disjoint subset of the demand nodes. This means that each demand node has exactly one arc with positive flow incident to it, so the number of arcs with positive flow is only N.

Since there are $M+N-1$ arcs in a spanning tree over $M+N$ nodes, and the addition of a single arc to a tree results n a cycle, then the interpolation process will

create cycles whenever $NM/2$ is greater than $M+N-1$, which will be true for any long transportation problem whenever $M>2$ and $N>3$. So, for most problems, the interpolation process will introduce cycles. In order to more fully understand this process, the $3\times N$ problem will be taken as an example. Some of the possible combinations produced in the $2\times N$ and $3\times N$ interpolation routines are illustrated in Figures 12 and 13.

In each subproblem of the $3\times N$ problem, either one, two or all three supply nodes may be providing flow to either of the demand nodes. In the case that only one supply node supplies the demand nodes, it is impossible for cycles to develop. Since there must be two arcs with positive flow in each problem, the union of two such subproblems, with common supply nodes, will have five nodes and four arcs with positive flow.

If there are two active supply nodes in the union of the two subproblems, then there will be three arcs with positive flow in a non-degenerate problem, and two arcs in a degenerate problem. For every two non-degenerate subproblems, a cycle will be created in the initial solution to the fine level problem, since there will be six nodes and six arcs with positive flow. The union of a degenerate problem and a non-degenerate problem will result in a non-degenerate basic feasible solution (six nodes and five arcs with positive flow) on Ω^k, while the union of two degenerate subproblems will result in a degenerate problem on the fine level (six nodes and four arcs).

In the case of three active supply nodes, the possibilities are either three or four arcs with positive flow for each subproblem. The union of two non-degenerate subproblems will result in two cycles being introduced on Ω^k. A non-degenerate and a

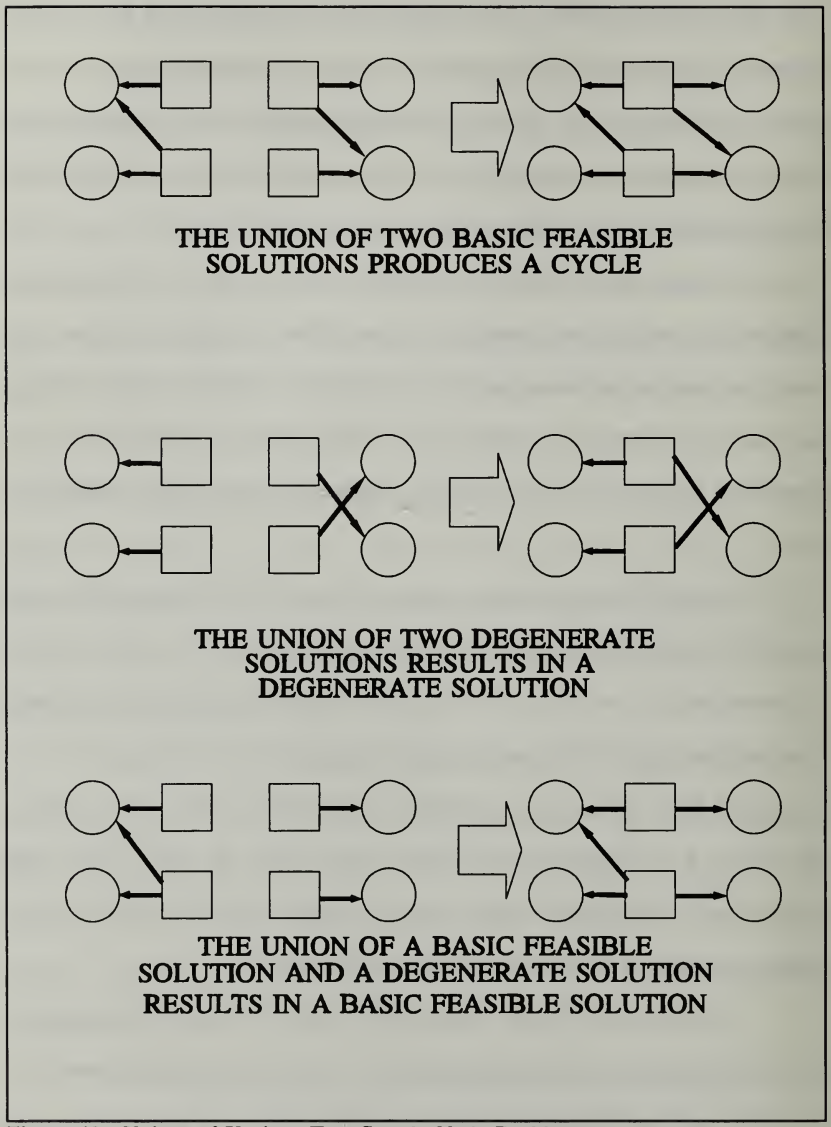
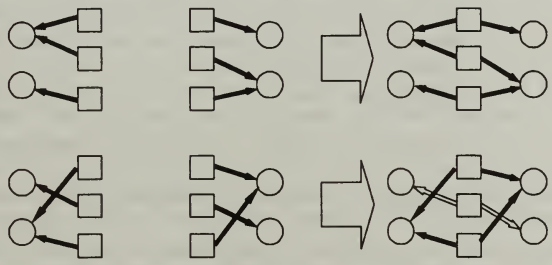

THE UNION OF TWO BASIC FEASIBLE
SOLUTIONS PRODUCES A CYCLE
THE UNION OF TWO DEGENERATE
SOLUTIONS RESULTS IN A
DEGENERATE SOLUTION
THE UNION OF A BASIC FEASIBLE
SOLUTION AND A DEGENERATE SOLUTION
RESULTS IN A BASIC FEASIBLE SOLUTION

Figure 12. Unions of Various Two Supply Node Problems

THE UNION OF TWO BASIC
FEASIBLE SOLUTIONS PRODUCES TWO CYCLES



THE UNION OF TWO DEGENERATE
SOLUTIONS PRODUCES EITHER A
BASIC FEASIBLE SOLUTION OR A CYCLE

THE UNION OF A DEGENERATE
SOLUTION AND BASIC SOLUTION PRODUCES
A SINGLE CYCLE

Figure 13. Unions of Three Supply Node Subproblems.

degenerate combination will beget a single cycle, and two degenerate subproblems may result in either a basic feasible solution or a degenerate solution containing a cycle on Ω^k.

These possible combinations apply whenever two subproblems share common supply nodes. Since the demand nodes of each subproblem are disjoint, cycles can only be created when two demand nodes from different subproblems share common supply nodes. Since a network flow problem is a special case of a linear programming problem, an optimal solution must exist at an extreme point that corresponds to a basic feasible solution. While this solution may in fact be degenerate, i.e., contain fewer than *M+N-1* arcs with positive flow, it cannot have more; that is, a solution with more than *M+N-1* positive flows is not an extreme point solution. A reasonable candidate for a local relaxation process is to adjust the flow in the initial solution produced by the interpolation process, so that cycles are removed and the objective function is reduced. The effect of this procedure is to adjust the locally optimal flows which result from interpolation so that they are more nearly optimal in the global problem.

Cycles are detected in the algorithm using a *depth first search (DFS)*. The DFS proceeds as follow:

1. Initialize all nodes with DFS number 0, to indicate they have not yet been visited.
2. Start at any node. In this particular case, the second demand node of the second subproblem is chosen to begin the search. Assign this node a DFS number of 1, and define node 0 to be the predecessor of this node.
3. If any node adjacent to the current node has been previously visited, and has a DFS number lower than the predecessor of the current node, then the path from that node through the current node and back is a cycle. Stop DFS and call the cycle removal routine.

4. If no adjacent nodes have lower DFS numbers, then look for any adjacent nodes which have not been visited. If there are any unvisited adjacent nodes, identify the current node as the predecessor of the unvisited node, make the unvisited node the current node, and assign the current node a DFS number equal to the DFS number of its predecessor plus 1.

5. If there are no unvisited nodes adjacent to the current node, make the predecessor of the current node the current node. If the current node is node 0, stop. Otherwise, return to step 3.

Once a cycle is detected, a cycle removal algorithm is used to adjust the flows. This algorithm uses a mechanism very similar to the one GNET uses when pivoting a new arc into the basis. This technique is illustrated in Figure 14. The unit costs of a change in flow in both the clockwise and counter clockwise directions around the cycle are determined by adding together the costs of the arcs whose flow increases and subtracting the cost of the arcs whose flow decreases. The change in objective function value per unit change in flow in one direction will be the negative of the change in the opposite direction. For example, in Figure 14, a unit increase in flow clockwise around the cycle will cause a change in the objective function value of 5-4+8-3=6, an increase. A unit increase in the counter-clockwise direction yields -6, a decrease of 6 in the objective function. Clearly, increasing the counter-clockwise flow is profitable, so flow is increased in this direction. Flow will thus be increased on arcs (S_1, D_2) and (S_2, D_1) and decreased on arcs (S_1, D_1) and (S_2, D_2), until the flow on (S_1, D_1) reaches zero. That is, flow is increased on the arcs in the cycle which point in the profitable direction, and decreased on the arcs which point against the flow, until one of the decreasing arcs reaches zero flow. At this point, the number of arcs in the cycle with positive flow has

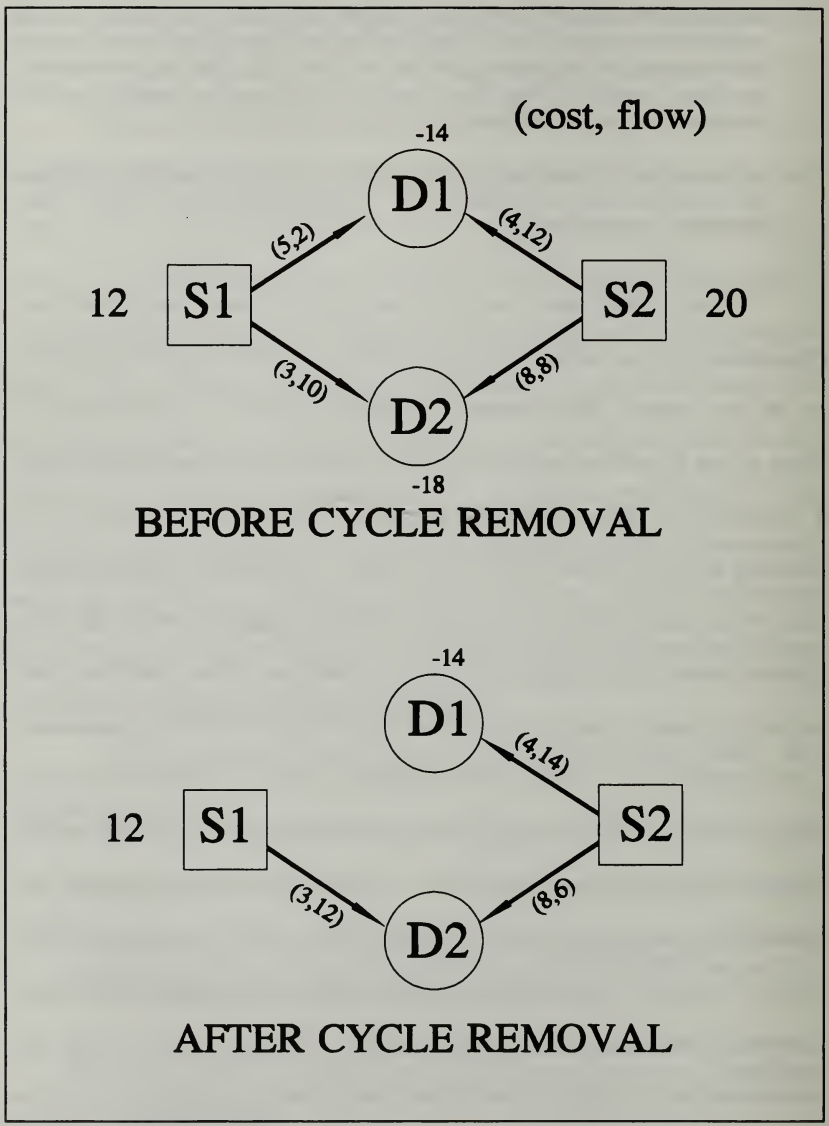


Figure 14. Cycle Removal

been reduced by one, the cycle has been removed, and the value of the objective function has been lessened from 152 to 140.

This technique is used as a local relaxation method by applying it to pairs of subproblems. Two subproblems which are adjacent in cost space are joined to form an $M\times4$ problem, which is inspected for cycles. If any are found, they are removed and the problem is searched again.

Two different methods for applying this technique were investigated. The first was to remove the cycles from adjacent $M\times4$ problems, then repeat the process by joining pairs of $M\times4$ problems, then $M\times16$, and so on until the global problem for the current level is inspected and certified cycle free. This approach was extremely inefficient. The second approach only considers more local changes. That is, only pairs of the interpolated subproblems were checked for cycles. The gain in speed from using this second method were tremendous, and the decrease in accuracy was negligible. These results are summarized in Table III.

While the multilevel algorithm performed well on problems with only two or three supply nodes, the results for the five supply node problem are unsatisfactory. The above table clearly indicates that relaxation by cycle removal is as effective when applied over a local area as when applied globally, and the computational effort required for local relaxation is an order of magnitude smaller.

Table III. CYCLE REMOVAL ALGORITHM

PROBLEM SIZE	SOLUTION METHOD	RUN TIME	% ABOVE OPTIMALITY
2 x 1024	COMPLETE RELAXATION	1.210835	.02
2 x 1024	LOCAL RELAXATION	.131437	.02
2 x 1024	GNET	.041132	0
3 x 1024	COMPLETE RELAXATION	1.15788	8.41
3 x 1024	LOCAL RELAXATION	.108765	8.41
3 x 1024	GNET	.076648	0
5 x 1024	COMPLETE RELAXATION	1.18411	58.4
5 X 1024	LOCAL RELAXATION	.161112	59.7
5 x 1024	GNET	.106392	0

VI. CONCLUSIONS AND RECOMMENDATIONS

Research into the applications of multilevel techniques to optimization problems is only beginning. Opportunity exists for much further research in this area. The work done in this thesis should provide some valuable insights, and point out some of the areas where the potential benefit is the most promising.

Of the schemes investigated, the best results to date have been obtained using the cycle removal algorithm for local relaxation in an FMG cycle, demand-weighted restriction and $M\times 2$ interpolation. The use of demand-weighted restriction and $M\times 2$ interpolation parallels the results obtained for the *geometrical* long transportation problem, (Kaminsky, 1979), but they differ significantly in the method of local relaxation. Also, Kaminsky used one method of interpolation from the end of a V-cycle on Ω^{k+1} to the start of a V-cycle on Ω^{k}, and another method within each V-cycle, whereas we use the same interpolation throughout.

The results obtained thus far, while providing some positive indications, do not show that a multilevel approach is as efficient a method as the primal network simplex. Network simplex algorithm represents 20 years of fine tuning and extensive computational research. It should not be expected that a new algorithm would show comparable results initially. The current results do show that this approach may be used in optimization, and may be used on a much larger class of problems than was previously thought. As improvements to the methodology are made, the multilevel approach may eventually

prove to be an effective alternative solution method. Many possibilities for improving the performance of the methods developed in this paper exist. Some of these are discussed briefly in the following sections. First, however, a summary of the highlights of the current research is given.

A. SIGNIFICANT RESULTS

The most significant contribution of the current research is the removal of the requirement for a physical interpretation of the problem, and the dependence on a relationship between distance and shipping costs. By mapping the problem into cost-space, a multilevel approach can be applied to a much broader class of problems. Of course, there is a limit to the number of supply nodes which this approach can handle, due to the increasing dimensionality of the problem. However, for problems with few supply nodes, this approach can be helpful. The result is that problems which have either a very small number of supply nodes, or a geometrical interpretation, can be solved to within an acceptable degree of optimality using a multilevel approach, however, problems which do not meet either of these criteria do not seem to be tenable to currently known multilevel methods.

The realization that the interpolation process would introduce cycles into the global problem is important for future research. Any algorithm which exploits the tree structure of a basic feasible solution should be improved by searching for and removing cycles.

Finally, the procedure for rapidly identifying the vector of dual multipliers (to within a constant, which is all that is required), may have important applications in a

future FAS algorithm. When a method for extracting the correction from the solution to the problem on the coarser level is found, then using the dual multipliers, which is analogous to using the residuals in a partial differential equation, may be the appropriate restriction method.

B. AVENUES OF FURTHER RESEARCH

First and foremost, an algorithm analogous to the full approximation scheme (FAS) should be more fully developed. In the current approach, we were unable to find an effective method of extracting a *correction* from the solution on Ω^{k+1} and applying it to the approximation on Ω^{k}, while still maintaining feasibility. Instead, we computed the solution on Ω^{k+1} and use interpolation to *replace* the solution on Ω^{k}. Since a direct analog to the residual in a PDE is difficult to obtain in an optimization problem, working with the original problem (as opposed to the residual equation) seems like a better approach. This would indicate that FAS is the method of choice, however, the difficulty mentioned above must be overcome.

Another possibility for improving the current algorithm is to begin the procedure by overlaying the cost-space with a regular M-dimensional grid (M-*1* in the 'reduced dimension' problem). The first step of the restriction process would then be to map the demand nodes from their natural irregularly spaced positions in cost-space to the regular grid points. Later, the final interpolation step would be to transfer from the regular grid back to the original demand points. This approach overcomes a shortcoming in the current algorithm, which aggregates demand nodes which are closest in relative distance

in cost-space, regardless of the absolute distance between them. In using a regular grid, a demand node on Ω^{k+1} would reflect only the demand at nodes within a small distance away, relative to the level $k+1$. Another important potential advantage is that the work on each coarser level is reduced by 2^{-M}, where M is the dimensionality of the problem. By contrast, the current algorithm reduces the work by approximately half in going to a coarser level.

If the regular grid approach proves worthwhile, then it could be extended to a *fast adaptive composite* (FAC) grid approach. In FAC, the single irregular grid is replaced by the union of two or more regular grids of different mesh spacing. That is, regions of the domain which benefit from a closer grid spacing are examined on a finer grid, while regions which do not require as much attention are mapped initially to a coarser grid. A V-cycle is started on the fine grid region, and when the level of coarseness of the remainder of the problem is reached, the domain is extended to include the entire problem.

The benefit of an FAC algorithm is that information from the sparse regions of the problem is exchanged with information concerning the dense regions while the algorithm is working on coarser levels. This information is then disseminated throughout the dense regions as the algorithm moves to finer and finer levels.

In a network optimization setting, this might be done by overlaying a fine grid on those regions of cost-space where the density of demand nodes is high, and a coarser grid on the areas of low density. In this way, the flow to nodes which are most similar to their nearest neighbors in cost-space will receive the benefit of a finer grid spacing, while

nodes which are naturally more distinct from their neighbors will only enter the problem on the coarser levels.

The current (FMG) algorithm will benefit from further research in at least two areas. The first is a graph-theoretic study of the interpolation and relaxation processes. The efficiency of the algorithm would definitely improve if a better cycle identification and removal routine is developed. Currently, every pairing of subproblems is searched for cycles, and, if one is found and removed, the search is restarted. One focus of a graph-theoretic study would be to identify those subproblems which introduce cycles into the global problem, forecast how many cycles will be created, and possibly even which nodes will be in the cycle, so that the cycles may be removed quickly.

The second area for potential improvement of the current algorithm is in developing more efficient data structures. The current approach is somewhat extravagant in its use of storage space, since costs, flows, and demands for each node at each level are stored in $M \times N \times \tilde{k}$ arrays. With adequate bookkeeping, this storage requirement could be reduced to $M \times N \times 2$ for each variable, which for large problems would be a significant reduction in storage space, and possibly in execution time.

APPENDIX

```
      PROGRAM FMG


C  WRITTEN BY KEVIN J. CAVANAUGH
C           1 NORMAN DRIVE
C           GALES FERRY, CT  06335
C
C  IN PARTIAL COMPLETION OF THE REQUIREMENTS FOR
C  MS IN OPERATIONS RESEARCH
C  MS IN APPLIED MATHEMATICS
C  NAVAL POSTGRADUATE SCHOOL
C  MONTEREY, CA
C  SEPTEMBER, 1992
C

C THIS PROGRAM WILL PERFORM A FULL MULTIGRID SCHEME ON A LONG
C TRANSPORTATION PROBLEM.
C THE MULTISTAGE APPROACH IS TO COMBINE DEMAND NODES BASED
C ON THEIR PROXIMITY IN COST SPACE.
C
C THESE SUPERNODES ARE THEN FURTHER COMBINED,  AND
C THIS COARSENING IS REPEATED UNTIL THE NUMBER OF DEMAND NODES
C IS EQUAL TO 1. AT THIS POINT THE OPTIMAL SOLUTION FOR THIS
C PROBLEM IS EASILY FOUND.
C
C THE SOLUTION TO THIS COARSE PROBLEM IS THEN INTERPOLATED TO
C THE NEXT FINER LEVEL.
C LOCAL RELAXATION IS PERFORMED ON THIS LEVEL BY REMOVING
C CYCLES CREATED BY THE INTERPOLATION PROCESS
C
C THE INTERPOLATION - OPTIMIZATION SEQUENCE IS REPEATED UNTIL
C THE FINEST (ORIGINAL) LEVEL IS ONCE AGAIN REACHED.

C     VARIABLE DECLARATIONS
      IMPLICIT INTEGER (A-Z)
      INTEGER RDA(7), LOOP, KK, NR,GROUP, OVERLAP,
     X FLOWIN(1100),
     X U(1100),X(1100),P(1100),DP(6000),IT(1100),
     X CPX(1100),NSA(1100),ISA(1100),A(1101),
     X LOC(1100), OUTFLOW(50), IN68, MR, DN(1100)
     X I,J, ACODE, BCODE, CSECODE, TOTAL_SOLVE_TIME
     X ,ACOST(0:10), BCOST(0:10), D1,D2, INCR,INCR2

      REAL*8 ATIME,BTIME,TEMP,CSETIME, TIMING(3,0:10),
     X  TEMPTIME
```

```
      INCLUDE 'COMMON BLOCK A'

      INCLUDE 'STRUCT BLOCK A'

C     INITIALIZE ARRAYS AND VARIABLES

      INFINITY = 1000000
      RINF = 1000000.0
      DO 20 I = 1,50
        SUPPLY (I) = 0
        DO 30 J = 1,1100
          DO 40 K = 0,10
            COST (I,J,K) = INFINITY
            CPRIME (I,J,K) = RINF
40        CONTINUE
30      CONTINUE
20    CONTINUE


      DO 50 I = 1,1100
        NODE(I)=I
        DO 60 K=0,10
          DEMAND(I,K) = 0
60      CONTINUE
50    CONTINUE
      TOTAL_SUPPLY = 0
      TOTAL_DEMAND = 0
      OPEN (UNIT=1)
      K=0
      INPUT=1
      OUTPUT = 6
      NOD = 1100
      NAD = 6000
      MXC = 2000

C     READ IN THE ORIGINAL NETWORK ARCS, COSTS, CAPACITIES,
C     SUPPLIES AND DEMANDS
      CALL SHARE
     1 (M,S,HEAD,TAIL,C,CP,X,CPX,P,DP,IT,U,NSA,ISA,A,BIGI,
     2  MAXC, ISUP, IPRT, NR,
     3  INPUT,OUTPUT,NOD,NAD,MXC)

      D(0) = M-S
      I=1
      DO 70 J=1,S
        SUPPLY(J) = X(J)
70    CONTINUE
      DO 71 J=S+1,M
        DEMAND(J-S,0) = -1*X(J)
```

```
71    CONTINUE
      TOTAL_SUPPLY = ISUP
      TOTAL_DEMAND = ISUP

C   INITIALIZE COST ARRAY FOR FINEST LEVEL
      DO 997 I = S+1,M
        DO 998 J = HEAD(I), HEAD(I+1)-1
          COST(TAIL(J),I-S,0) = REAL(C(J))
998     CONTINUE
997   CONTINUE
      CLOSE (UNIT = 1)


10    CONTINUE
C     START CLOCK
      CALL CPUTIME(ATIME,ACODE)
103   FORMAT (' START TIME: ',F10.0,' START CODE: ',I3)
104   FORMAT (' END CODE:   ',F10.0,' END CODE:   ',I3)



C     COARSEN UNTIL THERE IS ONLY ONE COARSE DEMAND NODE
      CALL  COARSEN
      DO 500 J = 1,S
        FLOW(J,NODE(1)) = REAL(SUPPLY(J))
500   CONTINUE

109   FORMAT (' TIME TO COARSEN:  ', F10.0, ' MICROSECONDS')
      CALL CPUTIME(CSETIME,CSECODE)

199   CONTINUE

        DO 940 I = 1,D(0)-2**K+1,2**K
          CALL INTERP(I)
940     CONTINUE
        K = K-1

 201    FORMAT (1X, 5F6.0)

C NR IS THE NUMBER OF RELAXATION SWEEPS, USUALLY 1.
C MR IS THE NUMBER OF DEMAND NODES IN THE SUBPROBLEMS BEING
C DE-CYCLED.
        CALL CPUTIME(TEMPTIME,ACODE)
```

```
      IF (K.LE.KMAX-2) THEN
        MR = 2
C IF COMPLETE RELAXATION IS DESIRED, UNCOMMENT THE NEXT LINE
C211      CONTINUE
          MR = 2*MR
          LP = 1
221         CONTINUE
            DO 231 I = 1, MR
              DN(I) = NODE(LP+(I-1)*(2**K))
231           CONTINUE
            CALL RELAX (S, MR, DN, FLOW, CPRIME,K)
            LP = LP +MR*(2**K)
          IF (LP.LT.D(0)) GO TO 221
C  F COMPLETE RELAXATION IS DESIRED, UNCOMMENT THE NEXT LINE
C       IF (MR.LT.D(K)) GOTO 211
        CALL CPUTIME(TIMING(3,K),ACODE)
        TIMING(3,K) = TIMING(3,K) - TEMPTIME
      ENDIF
3     FORMAT (5(I3,I5))
      KK = K


C     PERFORM A W CYCLE FROM THIS POINT
      DO 950 LOOP = KK,KMAX-1
        CALL WCOARSE(KK)
        K = K+1
950     CONTINUE
960     CONTINUE
        DO 965 I = 1,D(0)-2**(K-1),2**K
          CALL INTERP(I)
965       CONTINUE
        K =K - 1
      IF (K.GT.KK) GO TO 960
      CALL CPUTIME(TIMING(1,K),ACODE)
      TIMING(1,K) = TIMING(1,K) - TEMPTIME
 2     FORMAT (8F6.0)
    IF (K .GT. 0) GOTO 199

C     STOP CLOCK
    CALL CPUTIME(BTIME,BCODE)
    DO 146 I=1,S
      DO 127 J = 1,D(0)
        BCOST(K) = BCOST(K) + NINT(FLOW(I,J)*COST(I,J,0))
        OUTFLOW(I) = OUTFLOW(I)+FLOW(I,J)
127     CONTINUE
146   CONTINUE
    PRINT 109, CSETIME-ATIME
105   FORMAT (' ELAPSED CPU TIME:  ',F10.0, ' MICROSECONDS.')
    PRINT 105, BTIME-ATIME
106   FORMAT (' TOTAL COST: ', I19)
```

```
      PRINT *, 'TOTAL COST: ', BCOST(0)
      PRINT *
111   FORMAT (43H0  FROM  TO  FLOW    COST    DEMAND     )
112   FORMAT (3X,I3,2X,I4,3X,I5,8X, F5.0,5X,I4)
      WRITE (22,111)
      DO 113 J = 1,M
        FLOWIN(J) = 0
        DO 114 I = 1,S
          FLOWIN(J) = FLOWIN(J)+FLOW(I,J)
          IF (FLOW(I,J).GT.0) WRITE (22,112)
     &          I, J, FLOW(I,J), COST(I,J,0),DEMAND(J,0)
 114    CONTINUE
        IF (FLOWIN(J).NE.DEMAND(J,0)) PRINT *, 'INFEASIBLE.
     &      DEMAND NOT MET AT NODE', J
 113  CONTINUE
      WRITE (6,*)
      WRITE (6,*) ('SOURCE: ',K, ' TOTAL FLOW: ', OUTFLOW(K),
     &   K=1,S)
      STOP

      END
```

```
      SUBROUTINE WCOARSE(KK)

C THIS SUBROUTINE WILL PERFORM A WEIGHTED COARSENING ON THE
C DOWN SLOPE OF A VEE CYCLE.  DEMAND IS ALREADY COARSENED BY
C THE INITIAL UNWEIGHTED COARSENING.  COST IS WEIGHTED BY THE
C AMOUNT OF FLOW ON THE COMPONENT FINER LEVEL ARCS. FLOW ON
C THE FINER ARCS IS ADDED TO DETERMINE FLOW ON THE COARSE ARC

      INTEGER I,J,KK,D1,D2
      INCLUDE 'COMMON BLOCK A'
      INCLUDE 'STRUCT BLOCK A'

      INCR = 2**K

      DO 100 I = 1,D(0)-INCR+1,2*INCR
        D1 = NODE(I)
        D2 = NODE(I+INCR)
        DEMAND(D1,K+1) = DEMAND(D1,K) + DEMAND (D2,K)
        DO 110 J = 1,S
C   DEMAND WEIGHTED FLOW IS IMPLEMENTED HERE
          CPRIME (J,D1,K+1) = (DEMAND(D1,K)
     1          *CPRIME(J,D1,K) + DEMAND(D2,K)
     2          *CPRIME(J,D2,K))
     3          /REAL(DEMAND(D1,K+1))
          FLOW(J,D1)=FLOW(J,D1)+FLOW(J,D2)
          IF (FLOW(J,D1).LT.0) THEN
            PRINT *, 'ERROR--NEGATIVE FLOW, KK: ',KK,K,I
            PRINT *, FLOW(J,D1), FLOW(J,D2)
            PRINT *, 'ERROR DETECTED IN S/R WCOARSE'
            STOP
          ENDIF
          FLOW(J,D2) = 0
110     CONTINUE
100   CONTINUE


      RETURN
      END
```

```
      SUBROUTINE RELAX(S, M, DN, FLOW, CPRIME,K)
      LOGICAL FINISHED, FOUND, CYCLE
      INTEGER  TOP, FLOW(50,1100),
     1   PRED(1100), VISITED(1100), DN(1100),
     2   STACK(1100), NODETYPE, S, M, CURRENT
      REAL CPRIME(50,1100,0:10)

  100 CONTINUE
      CURRENT = DN(M)
      CYCLE = .FALSE.
      FINISHED = .FALSE.
      NODETYPE = 1

      DO 200 I = 1,M
        VISITED(DN(I)+S) = 0
        PRED(DN(I)+S) = 0
        STACK(I) = 0
  200 CONTINUE
      DO 210 I = 1,S
        VISITED(I) = 0
        PRED(I) = 0
  210 CONTINUE
      VISITED(CURRENT+S) = 1
      TOP = 0
  1   CONTINUE
C  CONDUCT A DEPTH FIRST SEARCH TO IDENTIFY CYCLE

C EXAMINE ARCS COMING OFF CURRENT NODE.  IF ANY NODES WITH     C LOWER
DFNUMBER ARE ADJACENT, THEN THERE IS A CYCLE.
        IF (NODETYPE.EQ.1) THEN
          I=1
          FOUND = .FALSE.
  10       CONTINUE
            IF (FLOW(I,CURRENT).GT.0) THEN
              IF (VISITED(I).EQ.0) THEN
                FOUND = .TRUE.
                PRED(I) = CURRENT
                VISITED(I) = VISITED(CURRENT+S)+1
                TOP = TOP+1
                STACK(TOP) = CURRENT
                CURRENT = I
                NODETYPE = 2
              ELSEIF (VISITED(I).LT.VISITED(CURRENT+S)-1)
     &             THEN
                CYCLE = .TRUE.
                CALL REMOVE (I, CURRENT, PRED, FLOW,
     &                 CPRIME, NODETYPE,K,S)
              ENDIF
            ENDIF
            I = I+1
```

```
        IF ((I.LE.S).AND.(.NOT. FOUND)) GOTO 10
      ELSE
        I = 1
        FOUND = .FALSE.
20       CONTINUE
          IF (FLOW(CURRENT,DN(I)).GT.0) THEN
            IF (VISITED(DN(I)+S).EQ.0) THEN
              FOUND = .TRUE.
              PRED(DN(I)+S) = CURRENT
              VISITED(DN(I)+S) =
     &         VISITED(CURRENT)+1
              TOP = TOP+1
              STACK(TOP) = CURRENT
              CURRENT = DN(I)
              NODETYPE = 1
            ELSE
              IF (VISITED(DN(I)+S).LT.
     &          VISITED(CURRENT)-1) THEN
                CYCLE = .TRUE.
                CALL REMOVE (DN(I), CURRENT,
     &            PRED, FLOW, CPRIME, NODETYPE,
     &            K, S)
              ENDIF
            ENDIF
          ENDIF
          I = I+1
        IF ((I.LE.M).AND.(.NOT. FOUND)) GOTO 20
      ENDIF
    IF (CYCLE) GOTO 100
    IF (.NOT.FOUND) THEN
      IF (TOP.GT.0) THEN
        CURRENT = STACK(TOP)
        NODETYPE = 3-NODETYPE
        TOP = TOP - 1
      ELSE
        FINISHED = .TRUE.
      ENDIF
    ENDIF

    IF (.NOT. FINISHED) GOTO 1
    RETURN
    END
```

```
      SUBROUTINE REMOVE (I, CURRENT, PRED, FLOW, CPRIME,
     &            NODETYPE,K,S)
      INTEGER CYCLE(100),IN,CRNT
     1   ,I, CURRENT, PRED(1100),S
     2   ,FLOW(50,1100), NODETYPE,K, DELTA, NTYPE
      REAL CPRIME(50,1100,0:10),CW
      NTYPE = NODETYPE
      CRNT = CURRENT
      CYCLE(1) = CRNT
      IN = 1
10    CONTINUE
        IN = IN+1
        IF (NTYPE.EQ.1) CYCLE(IN) = PRED(CRNT+S)
        IF (NTYPE.EQ.2) CYCLE(IN) = PRED(CRNT)
        NTYPE = 3-NTYPE
        CRNT = CYCLE(IN)
      IF ((CYCLE(IN).NE.I).OR.(NTYPE.EQ.NODETYPE)) GO TO 10
      IF (NODETYPE.EQ.1) THEN
        CW = CPRIME(CYCLE(IN),CYCLE(1),K)-CPRIME(CYCLE(IN),CYCLE(IN-1),K)
        DO 20 J = 2,IN-2,2
          CW = CW-CPRIME(CYCLE(J),CYCLE(J-1),K)+
     &          CPRIME (CYCLE(J),CYCLE(J+1),K)
20      CONTINUE
      ELSE
        CW = CPRIME(CYCLE(IN-1), CYCLE(IN),K) -
     &          CPRIME(CYCLE(1),CYCLE(IN),K)
        DO 30 J = 2,IN-2,2
          CW = CW+CPRIME(CYCLE(J-1),CYCLE(J),K)-
     &           CPRIME(CYCLE(J+1),CYCLE(J),K)
30      CONTINUE
      ENDIF
      IF (CW.LT.0.0) THEN
C     COMPUTE DELTA FLOW IN A CLOCKWISE TRAVERSAL
        IF (NODETYPE.EQ.1) THEN
          DELTA = FLOW(CYCLE(2),CYCLE(1))
          DO 40 J = 4,IN,2
            IF (FLOW(CYCLE(J), CYCLE(J-1)).LT.DELTA)
     &        DELTA = FLOW(CYCLE(J), CYCLE(J-1))
40        CONTINUE
          FLOW(CYCLE(IN),CYCLE(1)) = FLOW(CYCLE(IN), CYCLE(1))+DELTA
          FLOW(CYCLE(IN),CYCLE(IN-1)) =
     &      FLOW(CYCLE(IN),CYCLE(IN-1))-DELTA
          DO 50 J = 2,IN-2,2
            FLOW(CYCLE(J),CYCLE(J-1)) = FLOW(CYCLE(J),
     &           CYCLE(J-1)) -DELTA
            FLOW(CYCLE(J),CYCLE(J+1)) = FLOW(CYCLE(J),
     &           CYCLE(J+1)) + DELTA
50        CONTINUE
        ELSE
          DELTA = FLOW(CYCLE(1),CYCLE(IN))
```

```
      DO 60 J = 3,IN-1,2
        IF (FLOW(CYCLE(J),CYCLE(J-1)).LT.DELTA)
     &     DELTA = FLOW(CYCLE(J),CYCLE(J-1))
60     CONTINUE
      FLOW(CYCLE(1), CYCLE(IN)) = FLOW(CYCLE(1),
     &   CYCLE(IN) - DELTA
      FLOW(CYCLE(1), CYCLE(2)) = FLOW(CYCLE(1),CYCLE(2))
     &   + DELTA
      DO 70 J = 3,IN-1,2
        FLOW(CYCLE(J),CYCLE(J-1)) =
     &       FLOW(CYCLE(J),CYCLE(J-1)) - DELTA
        FLOW(CYCLE(J),CYCLE(J+1)) =
     &         FLOW(CYCLE(J),CYCLE(J+1)) + DELTA                    70
CONTINUE
    ENDIF
   ELSE
C  COMPUTE DELTA FLOW IN A COUNTERCLOCKWISE TRAVERSAL
    IF (NODETYPE.EQ.1) THEN
      DELTA = FLOW(CYCLE(IN),CYCLE(1))
      DO 45 J = 2,IN-2,2
        IF (FLOW(CYCLE(J), CYCLE(J+1)).LT.DELTA)
     &     DELTA = FLOW(CYCLE(J), CYCLE(J+1))
45     CONTINUE
      FLOW(CYCLE(IN),CYCLE(1)) = FLOW(CYCLE(IN),
     &    CYCLE(1)) - DELTA
      FLOW(CYCLE(IN),CYCLE(IN-1)) =
     &    FLOW(CYCLE(IN),CYCLE(IN-1))+DELTA
      DO 55 J = 2,IN-2,2
        FLOW(CYCLE(J),CYCLE(J-1)) = FLOW(CYCLE(J),
     &       CYCLE(J-1)) +DELTA
        FLOW(CYCLE(J),CYCLE(J+1)) = FLOW(CYCLE(J),
     &       CYCLE(J+1)) - DELTA
55     CONTINUE
    ELSE
      DELTA = FLOW(CYCLE(1),CYCLE(2))
      DO 65 J = 3,IN-1,2
        IF (FLOW(CYCLE(J),CYCLE(J+1)).LT.DELTA)
     &       DELTA = FLOW(CYCLE(J),CYCLE(J+1))
65     CONTINUE
      FLOW(CYCLE(1), CYCLE(IN)) = FLOW(CYCLE(1),
     &    CYCLE(IN)) + DELTA
      FLOW(CYCLE(1), CYCLE(2)) = FLOW(CYCLE(1),
     &    CYCLE(2))  - DELTA
      DO 75 J = 3,IN-1,2
        FLOW(CYCLE(J),CYCLE(J-1)) =
     &      FLOW(CYCLE(J),CYCLE(J-1)) + DELTA
        FLOW(CYCLE(J),CYCLE(J+1)) =
     &      FLOW(CYCLE(J),CYCLE(J+1)) - DELTA
75     CONTINUE
    ENDIF
```

```
      ENDIF
      RETURN
      END
      SUBROUTINE INTERP (INNODE)

      INCLUDE 'COMMON BLOCK A'

      INCLUDE 'STRUCT BLOCK A'

      REAL DIFF(50), TD
      INTEGER  FLOWIN(50), FLOWOUT(50), RS(50), UD1,UD2

      INTEGER  I, J, JJ ,D1,D2, IN, TJ, JOINT(50), KM, INNODE

C     COMMON /RELX/ RS, UD1, UD2

      KM = K-1
      D1=NODE(INNODE)
      D2 = NODE(INNODE+2**KM)
      DO 100 J = HEAD(S+D1),HEAD(S+D1+1)-1
        RS(TAIL(J)) = FLOW(TAIL(J),D1)
        FLOWIN(TAIL(J)) = FLOW(TAIL(J),D1)
100   CONTINUE
      UD1= DEMAND(D1,KM)
      UD2= DEMAND(D2,KM)
      IN = 1
      J  = 1
      JJ  = 1

C     FIRST, SEE IF ANY SUPPLIERS FEED ONLY ONE FINE NODE

      DO 115 J = HEAD(S+D1),HEAD(S+D1+1)-1
        IF (CPRIME(TAIL(J),D1,KM).GE.INFINITY-10) THEN
          UD2 = UD2 - RS(TAIL(J))
          FLOW(TAIL(J),D2) = RS(TAIL(J))
          IF (FLOW(TAIL(J),D2).LT.0) THEN
            PRINT *, ' ERROR -- NEGATIVE FLOW!  FLOW IS ',
     &       FLOW(TAIL(J),D2), 'K: ',K,I
            PRINT *, 'ERROR DETECTED IN S/R INTERP'
            STOP
          ENDIF
          RS(TAIL(J)) = RS(TAIL(J))-UD2
        ELSEIF (CPRIME(TAIL(J),D2,KM).GE.INFINITY - 10) THEN
          UD1 = UD1 - RS(TAIL(J))
          FLOW(TAIL(J),D1) = RS(TAIL(J))
          IF (FLOW(TAIL(J),D1).LT.0.0)THEN
            PRINT *, ' ERROR -- NEGATIVE FLOW!  FLOW IS ',
     &       FLOW(TAIL(J),D1), 'K: ',K,I
            PRINT *, 'ERROR DETECTED IN S/R INTERP'
            STOP
```

```
          ENDIF
          RS(TAIL(J)) = RS(TAIL(J))-UD1
        ELSE
          JOINT(JJ) = TAIL(J)
          JJ = JJ+1
        ENDIF
 115  CONTINUE
      JJ= JJ-1

C     NOW WORK ON DIVIDING SHARED FLOWS

      DO 150 J = 1, JJ
        DIFF(J) = ABS(CPRIME(JOINT(J),D1,KM)
     &         -CPRIME(JOINT(J),D2,KM))
 150  CONTINUE
C     SORT SUPPLY NODES BY INCREASING DIFFERENCE IN SHIPPING COSTS
      DO 160 J = 1,S-1
        DO 170 IN = J+1,S
          IF (DIFF(IN).GT.DIFF(J)) THEN
            TJ = JOINT(IN)
            TD = DIFF(IN)
            JOINT(IN) = JOINT(J)
            DIFF(IN) = DIFF(J)
            JOINT(J) = TJ
            DIFF(J) = TD
          ELSEIF (DIFF(IN).EQ.DIFF(J)) THEN
            IF (MIN(CPRIME(IN,D1,KM),CPRIME(IN,D2,KM)).GT.
     &           MIN(CPRIME(J,D1,KM),CPRIME(J,D2,KM))) THEN
              TJ = JOINT(IN)
              TD = DIFF(IN)
              JOINT(IN) = JOINT(J)
              DIFF(IN) = DIFF(J)
              JOINT(J) = TJ
              DIFF(J) = DJ
            ENDIF
          ENDIF
 170    CONTINUE
 160  CONTINUE
C     DISTRIBUTE THE AVAILABLE SUPPLIES TO MEET DEMANDS
      J=1
      IF (JJ.GT.0) THEN
 200  CONTINUE
        IF(CPRIME(JOINT(J),D1,KM).LT.CPRIME(JOINT(J),D2,KM)) THEN
          IF (RS(JOINT(J)).GT.UD1) THEN
            RS(JOINT(J)) = RS(JOINT(J)) - UD1
            FLOW(JOINT(J),D1) = UD1
            IF (FLOW(JOINT(J),D1).LT.0)THEN
              PRINT *, ' ERROR -- NEGATIVE FLOW!  FLOW IS ',
     &          FLOW(JOINT(J),D1),'K: ',K,I
              PRINT *, 'ERROR DETECTED IN S/R INTERP'
```

```
          STOP
        ENDIF
        UD1 = 0
      ELSE
        UD1 = UD1 - RS(JOINT(J))
        FLOW(JOINT(J),D1) = RS(JOINT(J))
        IF (FLOW(JOINT(J),D1).LT.0)THEN
        PRINT *, ' ERROR -- NEGATIVE FLOWI  FLOW IS ',
&          FLOW(JOINT(J),D1),'K: ',K,I
        PRINT *, 'ERROR DETECTED IN S/R INTERP'
         STOP
        ENDIF
        RS(JOINT(J)) = 0
      ENDIF
      IF (RS(JOINT(J)).GT.UD2) THEN
        RS(JOINT(J)) = RS(JOINT(J)) - UD2
        FLOW(JOINT(J),D2) = UD2
        IF (FLOW(JOINT(J),D2).LT.0)THEN
          PRINT *, ' ERROR -- NEGATIVE FLOWI  FLOW IS ',
&           FLOW(JOINT(J),D2),'K: ',K,I
          PRINT *, 'ERROR DETECTED IN S/R INTERP'
          STOP
        ENDIF
        UD2= 0
      ELSE
        UD2 = UD2 - RS(JOINT(J))
        FLOW(JOINT(J),D2) = RS(JOINT(J))
        IF (FLOW(JOINT(J),D2).LT.0)THEN
          PRINT *, ' ERROR -- NEGATIVE FLOWI  FLOW IS ',
&           FLOW(JOINT(J),D2),'K: ',K,I
          PRINT *, 'ERROR DETECTED IN S/R INTERP'
          STOP
        ENDIF
          RS(JOINT(J)) = 0
        ENDIF
      ELSE
        IF (RS(JOINT(J)).GT.UD2) THEN
          RS(JOINT(J)) = RS(JOINT(J)) - UD2
          FLOW(JOINT(J),D2) = UD2
          IF (FLOW(JOINT(J),D2).LT.0)THEN
            PRINT *, ' ERROR -- NEGATIVE FLOWI  FLOW IS ',
&             FLOW(JOINT(J),D2),'K: ',K,I
            PRINT *, 'ERROR DETECTED IN S/R INTERP'
            STOP
          ENDIF
          UD2 = 0
        ELSE
          UD2 = UD2 - RS(JOINT(J))
          FLOW(JOINT(J),D2) = RS(JOINT(J))
          IF (FLOW(JOINT(J),D2).LT.0)THEN
```

```
          PRINT *, ' ERROR -- NEGATIVE FLOW!  FLOW IS ',
&           FLOW(JOINT(J),D2),'K: ',K,I
          PRINT *, 'ERROR DETECTED IN S/R INTERP'
          STOP
        ENDIF
        RS(JOINT(J)) = 0
        ENDIF
        IF (RS(JOINT(J)).GT.UD1) THEN
          RS(JOINT(J)) = RS(JOINT(J)) - UD1
          FLOW(JOINT(J),D1) = UD1
          IF (FLOW(JOINT(J),D1).LT.0)THEN
            PRINT *, ' ERROR -- NEGATIVE FLOW!  FLOW IS ',
&             FLOW(JOINT(J),D1),'K: ',K,I
            PRINT *, 'ERROR DETECTED IN S/R INTERP'
            STOP
          ENDIF
          UD1= 0
        ELSE
          UD1 = UD1 - RS(JOINT(J))
          FLOW(JOINT(J),D1) = RS(JOINT(J))
          IF (FLOW(JOINT(J),D1).LT.0)THEN
            PRINT *, ' ERROR -- NEGATIVE FLOW!  FLOW IS ',
&             FLOW(JOINT(J),D1),'K: ',K,I
            PRINT *, 'ERROR DETECTED IN S/R INTERP'
            STOP
          ENDIF
          RS(JOINT(J)) = 0
        ENDIF
      ENDIF
    SAVEX(J,D1) = FLOW(J,D1)
    SAVEX(J D2) = FLOW(J,D2)
    J=J+1
  IF (J.LE.JJ) GO TO 200
ENDIF
RETURN
END
```

```
      SUBROUTINE COARSEN

      INTEGER I,J,L,DMIN, ZERO, START, STOP, TEMP,INCR,NN,
     &    SUPNODE,DMNDNODE,ONE,II, D1, D2, INCR2, TC(50,1100)
      INCLUDE 'COMMON BLOCK A'


C     COMPUTE C PRIMED VALUES.  IT IS ASSUMED THAT SUPPLY NODE S
C     IS CONNECTED TO EVERY DEMAND NODE.
C     C PRIMED WILL BE EQUAL TO THE ORIGINAL COST MINUS THE COST
C     OF SHIPPING TO THE SAME DEMAND NODE FROM SUPPLY NODE S.
462   FORMAT (2H 1,2X,I3,2X,F10.0)
463   FORMAT (2H 2,2X,I3,2X,F10.0)
      DO 460 J = 1,D(0)
        DO 450 I = 1,S-1
          CPRIME(I,J,0) = COST(I,J,0) - COST(S,J,0)
          IF (COST(I,J,0).EQ.INFINITY) CPRIME(I,J,0) = RINF
          TC(I,J) = CPRIME(I,J,0)*DEMAND(J,0)
450     CONTINUE
        CPRIME (S,J,0) = 0.0
460   CONTINUE
      L=1
      START = 1
      STOP = D(0)
C     SORT VERTICES IN ASCENDING ORDER BASED ON DISTANCE FROM
C     EACH SUPPLY POINT IN TURN USING SHELL SORT
      NN=STOP
100   CONTINUE
        INCR = NN/2
25      CONTINUE
          DO 35 I = START+INCR,STOP
            J=I-INCR
38          CONTINUE
              IF (TC(L,NODE(J)).GT.TC(L,NODE(J+INCR)))
     &          THEN
                TEMP = NODE(J+INCR)
                NODE(J+INCR) = NODE(J)
                NODE(J) = TEMP
                J=J-INCR
              ELSE
                J = 0
              ENDIF
              IF (J.GE.START) GOTO 38
35          CONTINUE
            INCR = INCR/2
          IF (INCR.GT.0) GOTO 25
        IF (STOP .LT. D(0)) THEN
          START = STOP + 1
          STOP = MIN0(D(0),STOP + NN)
        ELSE
```

```
        START = 1
        STOP = D(0)/(2**L)
        NN=STOP
        L=L+1
      ENDIF
    IF (L .LE. S-1) GOTO100
    INCR = 1
999  CONTINUE
      KM = K
      K=K+1
      INCR2 = INCR
      INCR = INCR*2
      DO 50 I=1,D(0)-INCR2,INCR
        D1 = NODE(I)
        D2 = NODE(I+INCR2)
        DEMAND(D1,K) = DEMAND(D1,KM)
   &      +DEMAND(D2,KM)
        DO 55 J=1,S
          CPRIME(J,D1,K) = (CPRIME(J,D1,KM)+CPRIME(J,D2,KM))/2.0
55       CONTINUE
50     CONTINUE
      D(K)=(D(KM)+1)/2
    IF (D(K).GT.1) GOTO 999
    KMAX=K
    RETURN
    END
```

```
C   FILE COMMON BLOCK A

      INTEGER D(0:10),S,DEMAND(1100,0:10), SUPPLY(50), K, CP(6000),
     & NODE(1100), ANSX (7000), CCOST, KMAX, N0,
     & TOTAL_SUPPLY, TOTAL_DEMAND, INFINITY,
     & STEP, PVT1LM, PVT2LM, PVT1CT(0:10), PVT2CT(0:10),
     & INFEAS, M, N(0:10), FLOW(50,1100)
     & ,SAVEX(50,1100)

      REAL COST (50,1100,0:10), CPRIME(50,1100,0:10), RINF

      DOUBLE PRECISION SOLVETIME(0:10),STRTIME(0:10), STPTIME(0:10)

      COMMON /VARS/
     & D, S, DEMAND, SUPPLY, K, COST, INFINITY, STPTIME,STRTIME,
     & SOLVETIME, NODE, TOTAL_SUPPLY, TOTAL_DEMAND, ANSX, CCOST,      & STEP,
PVT1LM, PVT2LM, PVT1CT, PVT2CT, INFEAS, M, N,CP,KMAX,
     & CPRIME, FLOW, RINF, SAVEX, N0


C  FILE STRUCT BLOCK A

      INTEGER HEAD(1100), TAIL(6000), C(6000)

      COMMON /STRUCTURES/ HEAD,TAIL,C
```

LIST OF REFERENCES

Aho, A. V., Hopcroft, J.E., and Ullman, J.D., *Data Structures and Algorithms*, p. 290, Addison-Wesley Publishing Co., 1987.

Bazaraa, M.S., Jarvis, J.J., and Sherali, H.D., *Linear Programming and Network Flows*, pp. 425-578, John Wiley and Sons, 1990.

Bradley, G.H., Brown, G.G., and Graves, G.W., "Design and Implementation of Large Scale Primal Transshipment Algorithms", *Management Science, 24*, 1 (September 1977), 1-31.

Brandt, A., *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics*, pp. 7-90, GMD-AIW, 1984.

Brandt, A., "Introduction -- Levels and Scales", in D.J. Paddon and H. Holstein (eds.): *Multigrid Methods for Integral and Differential Equations*, p. 8, Clarendon Press, 1985.

Brandt, A., Ron, D., Amit, D. J., "Multi-level Approaches to discrete-state and stochastic problems", in W. Hackbusch and U. Trottenberg (eds.): Multigrid Methods II (Proceedings, Cologne 1985), *Lecture Notes in Math.*, 1228, Springer - Verlag.

Briggs, W. L., *A Multigrid Tutorial*, pp 1-59, Society for Industrial and Applied Mathematics, 1987.

Kaminsky, R., *Multilevel Solution of the Long Transportation Problem*, M.Sc. Thesis, The Weizmann Institute of Science, 1986.

Klingman, D., Napier, A., and Stutz, J., "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation , and Minimum Cost Network Problems", *Management Science, 20*, 5 (January 1974), 814-821.

Nemhauser, G. L., and Wolsey, L. A., *Integer and Combinatorial Optimization*, pp. 76-77, John Wiley and Sons, 1988.

Ron, D., *Development of Fast Numerical Solvers for Problems in Optimization and Statistical Mechanics*, Ph.D. Thesis, The Weizmann Institute of Science, 1987.

Zipkin, P.H., "Bounds on the Effects of Aggregating Variables in Linear Programs", *Operations Research, 28* (1980), 4, 903-916.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Professor Van Emden Henson, Code MA/Hv Department of Mathematics Naval Postgraduate School Monterey, CA 93943-5000	2
4.	Professor Richard E. Rosenthal, Code OR/Rl Department of Operations Research Naval Postgraduate School Monterey, CA 93943-5000	2
5.	Professor Jeffrey Leader, Code MA/Le Department of Mathematics Naval Postgraduate School Monterey, CA 93943-5000	1
6.	LCDR Kevin J. Cavanaugh, USCG 1 Norman Drive Gales Ferry, CT 06335	4

816-655

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

DUDLEY KNOX LIBRARY
3 2768 00018859 3